#86 December 1983

$2.95 ($3.50 in Canada)

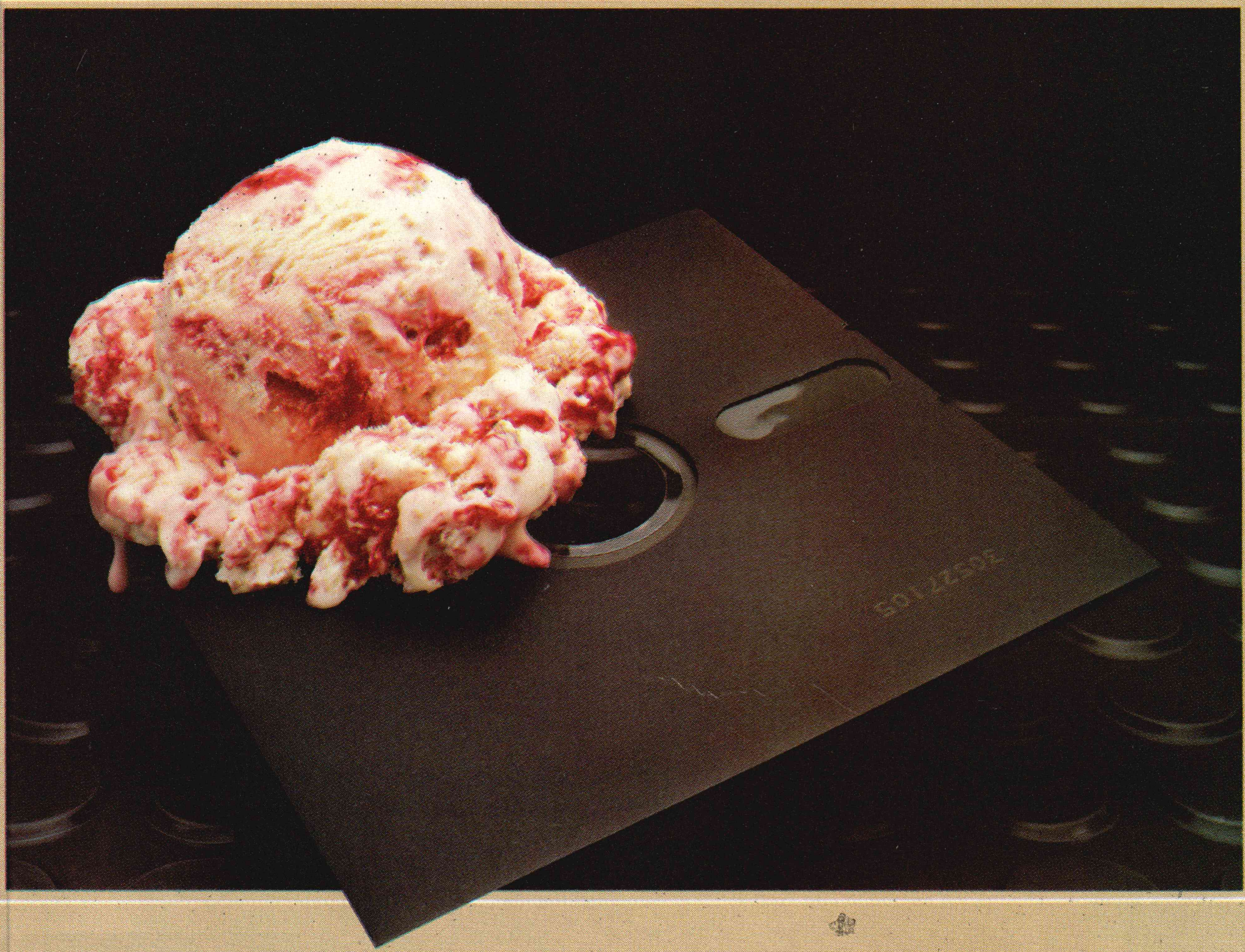
Dr. Dobb's Journal

For Users of Small Computer Systems

Faster Circles For Apples

Dysan's Digital Diagnostic Diskette

Cursor Control For Dumb Terminals



FAMOUS GIANT KILLERS THROUGHOUT HISTORY

David, Jack & PAT™

Size has always been a most formidable foe. Tales of David and Goliath and Jack and the Beanstalk are clear examples of how "Bigger is not always better". In fact, throughout history, all great giant killers have had one thing in common... small size but a surprisingly big capability.

And now, there's PAT, Dysan's unique Performance and Alignment tester that cuts giant floppy drive problems down to size.

PAT is the first hand-held intelligent tester for flexible diskette drives. Used in conjunction with Dysan's Digital Diagnostic Diskette (DDD)™, PAT's versatile and precise testing capabilities can test drive alignment and performance criteria. Plus, PAT also checks read/write data handling functions and serves as an exerciser without the use of additional equipment, all with minimal operator instruction.

And best of all, PAT sells for a small fraction of what the big testers cost.

PAT's Giant-Size Capabilities Include:

- Head radial alignment
- Head positioner linearity
- Head azimuth alignment
- Index timing
- Head positioner skew
- Spindle speed
- Read/write verification
- Seek test verification

For quality assurance, field service, incoming inspection, engineering evaluation and field disk drive sales, discover PAT. Another big difference from Dysan.

Dysan CORPORATION

Corporate Headquarters
5201 Patrick Henry Drive
Santa Clara, CA 95050
(800) 551-9000
(408) 988-3472

For more information on PAT, call toll free (800) 551-9000 or (408) 988-3472. Ask about our limited-time bonus offer!

Circle **no. 35** on reader service card.

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

Donating Subscribers **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

Contributing Subscribers: Christine Bell, W. D. Rausch, DeWitt S. Brown, Burks A. Smith, Robert C. Luckey, Transdata Corp., Mark Ketter, John W. Campbell, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Thomas Davis, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor, Jr., Ted A. Reuss III, Infoworld, Stan Veit, Western Material Control, S. P. Kennedy, John Hatch, Richard Jorgensen, John Boak. **Lifetime Subscriber:** Michael S. Zick.

Foreign Distributors **UK & Europe:** Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – Premiere, F-75014 Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** ASCII Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. *(Write for Canadian distributors)*

Dr. Dobb's Journal

For Users of Small Computer Systems

December 1983
Volume 8, Issue 12

CONTENTS

ARTICLES

DEPARTMENTS

Dr. Dobb's Journal (USPS 307690) is published twelve times per year by People's Computer Company, P.O. Box E, Menlo Park, CA 94026. Second class postage paid at Menlo Park, California 94026 and additional entry points. Address correction requested. Postmaster: send form 3579 to People's Computer Company, Box E, Menlo Park, CA 94026 · 415/323-3111 ISSN 0278-6508

WORKMAN & ASSOCIATES

112 Marion Avenue
Pasadena, CA 91106
(213) 796-4401

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing, and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. WRITE is $239.00.

"A PRIMER ON PASCAL FOR CP/M SYSTEMS"

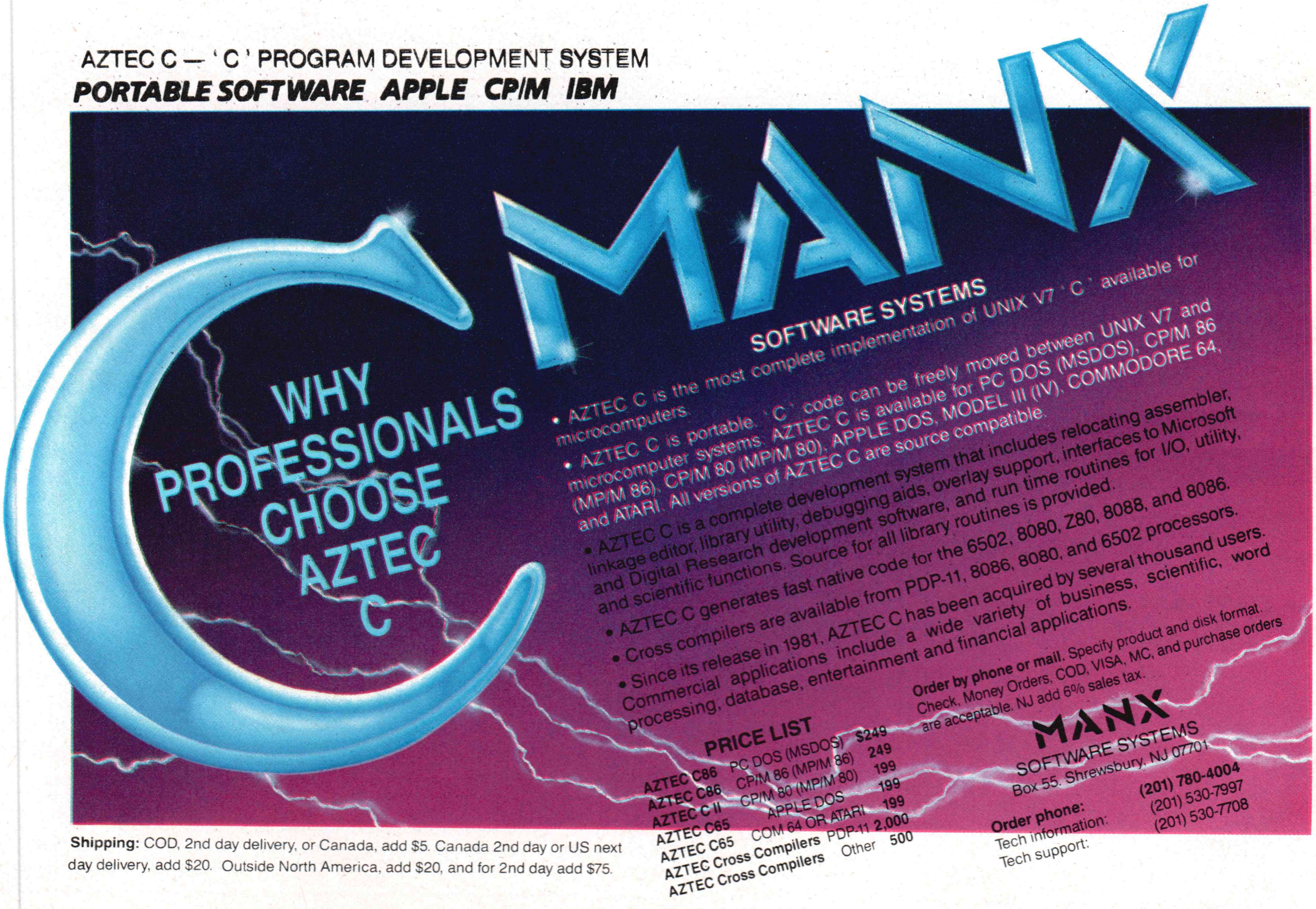
If you want to learn Pascal/M or MT+ on your own, this is the way. The Primer contains sample programs, suggestions, and notes on disk and in the manual. Programs come from the books "Programming in Pascal" and "Software Tools in Pascal". Discounted to only $64.50!

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, NorthStar, Osborne, KayPro, Monroe, Otrona Epson QX-10, NEC/PC,DEC VT-180, TI Professional, Access,Morrow. Please request our new catalog. We welcome COD orders.

Circle **no. 128** on reader service card.

AZTEC C — 'C' PROGRAM DEVELOPMENT SYSTEM

PORTABLE SOFTWARE APPLE CP/M IBM



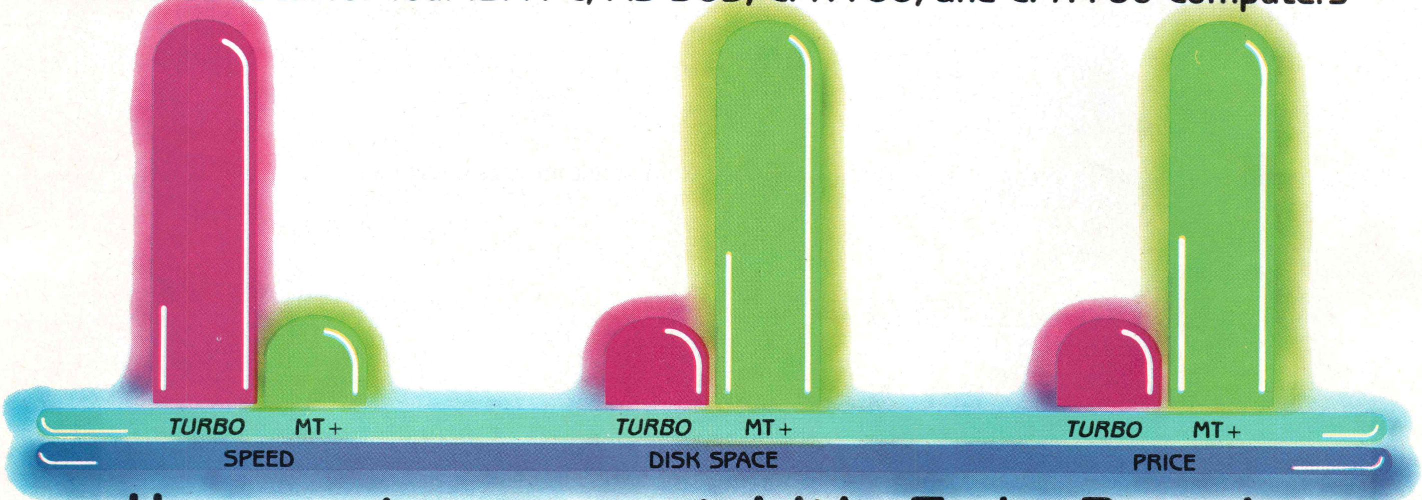
Shipping: COD, 2nd day delivery, or Canada, add $5. Canada 2nd day or US next day delivery, add $20. Outside North America, add $20, and for 2nd day add $75.

Circle **no. 61** on reader service card.

PASCAL $49.95

Introductory Offer

Standard Pascal for Your IBM PC, MS-DOS, CP/M 86, and CP/M 80 Computers



Hang on to your seats! It's *Turbo* Pascal.

	Turbo Pascal	JRT	MT+
8 & 16 bit	YES	NO	YES
Editor	YES	NO	NO
Generate Object Code	YES	NO	YES
Locates Run Time Errors Directly in Source Code	YES	NO	NO
Compilation Speed *	1 s.	46 s.	69 s. includes linking
Execution Speed *	6 s.	69 s.	8 s.
Disk Space	28K including editor	85K + editor	168K + editor
Price	$49.95	$39.95	$595.00

Benchmark data based on EightQueens in "Algorithms + Data Structures = Programs" by N. Wirth (Prentice-Hall, publisher)

Turbo Pascal is a trademark of Borland International. MT+ is a trademark of MT MicroSystems. JRT Pascal is a product of JRT. Wordstar is a trademark of Micropro.

There has never been a Pascal compiler this good with so many powerful features. We know what you've been waiting for: a true Pascal compiler that works fast, offers a full screen editor, and has a great price.

***Turbo* Pascal** has it all. First, we've included a built-in, interactive full screen, Wordstar compatible editor; it not only lets you correct errors, but during program compilation the cursor even jumps *directly* to the error and waits for your correction. No kidding. Second, it takes only 28K of disk space, including the editor; and on your microcomputer you need all the space you can get. *Turbo* Pascal is 10 to 70 times faster during compilation, as well as execution than Digital Research's MT+ or JRT Pascal.

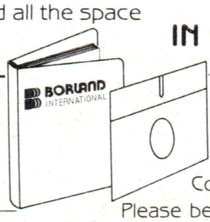
Hard to believe your good fortune on the price? Don't worry. We're Borland, and we produce only quality, state-of-the-art software. Companies such as Micro Pro, Morrow Computers, Access and others distribute our software products, so you can't go wrong.

Place your order today. And we'll ship your *Turbo* Pascal out fast. For VISA and MasterCard orders call toll free:

1-800-227-2400 X 968
IN CA: 1-800-772-2666 X 968

***Turbo* Pascal $49.95 + $2.00 shipping per copy.**

Check ___ Money Order ___
VISA ___ MasterCard ___
Card #: ____________
Exp date: _____ Shipped UPS.



My system is: 8 bit _____ 16 bit _____
Operating system: CP/M 80 ___
CP/M 86 ___ MS DOS ___ PC DOS ___
Computer: _________ Disk Format: _________
Please be sure model number and format are correct.

NAME: ____________
ADDRESS: ____________
CITY/STATE/ZIP: ____________
TELEPHONE: ____________

California residents add 6½% sales tax. Outside North America add $15.00 for airmail, or $5.00 for surface mail. Checks must be on a U.S. bank, and in U.S. dollars. Sorry, no C.O.D.

BORLAND INTERNATIONAL

Borland International
4807 Scotts Valley Drive
Scotts Valley, California 95066

Dealer and Distributor inquiries welcome.

Q: What's easier than spending money?

A: Keeping track of it with... Checks & Balances

At last! A *full screen* editing checking and finance program so informative and easy to use you'll wonder how you lived without it! *Especially* at **$49.95**

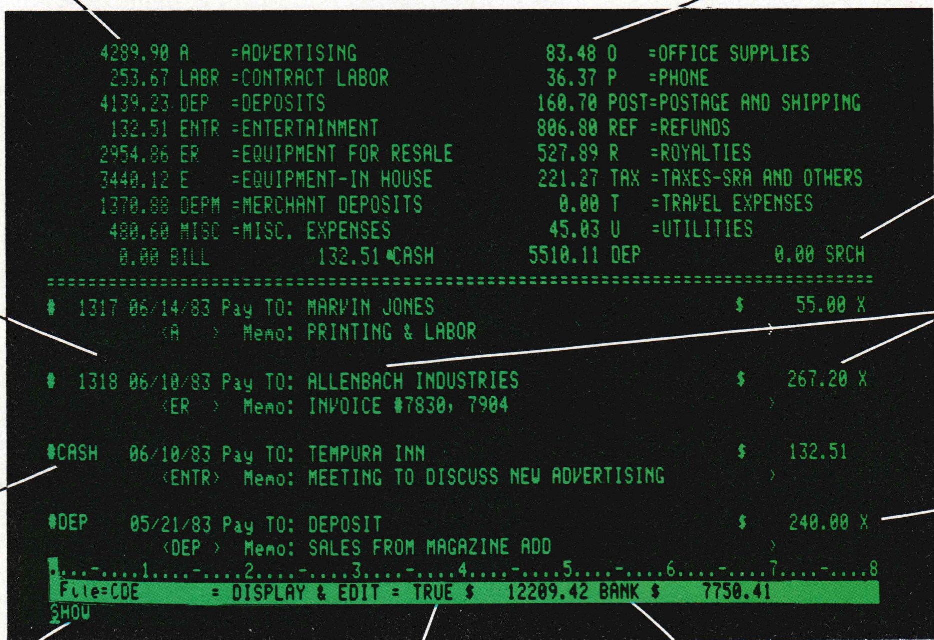
Sixteeen user-definable categories.

Category totals instantly updated with each new entry or change.

Includes categories for bills, cash expenditures, deposits and service charges.

Screen always shows four entries. Use your cursor to add or change any entry anytime. Scroll forward and back through your register or instantly go to any entry.

Checks and Balances aligns decimals and left justifies text.



For complete records, cash expenditures are included in category totals but not bank balance.

Entries which have cleared through your bank.

All commands in plain English.

True checking balance instantly updates with each entry.

Bank statement balance reflects only items checked off as having cleared your bank.

So easy to use! See your financial totals start to add up before your eyes in less than five minutes! Checks and Balances' cursor can move anywhere, anytime, to add or change entries, and all totals will instantly be updated on your screen when you change an amount or move a check to a different category. Checks and Balances gives you complete printed reports of the entire register, each category, or reports based on any entry in "Pay to" or "Memo"—even bills and uncashed or missing checks. File size is limited only by your disk capacity. Over 32,000 entries are possible.

Technical: 80 × 24 CRT with addressable cursor, reverse video optional. Z-80 processor, one or more disk drives with over 180k capacity. 56k RAM.

VISA®

CDE SOFTWARE

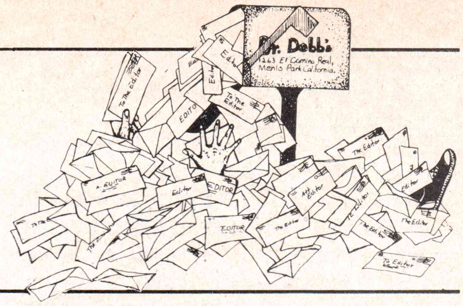
2463 McCready Avenue
Los Angeles, CA 90039

MasterCard

(213) 661-2031

Circle **no. 14** on reader service card.

LETTERS



Augusta Q & A

Dear *DDJ*:

Edward Mitchell's four-part series on his Augusta compiler, concluded in the July 1983 issue, was certainly interesting. The sheer magnitude of effort that he put into it is mind-boggling. It was definitely a plus that he gave a detailed description of his program, rather than just a long listing.

But I find myself baffled by his penchant for doing things the hard way. He rightly points out that BASIC is a less than ideal language for compiler writing. OK, he couldn't find a version of Pascal, C, or Forth to run on his Osborne. (Really??) But since he's obviously interested in systems programming, why on earth use an Osborne?

Along the same lines, one wonders why he chose to use a recursive descent parser. When coded in a language that permits recursive subroutine calls, one can write such a parser simply, rapidly, and bug-free. But BASIC permits no such thing. Since he had to go to the trouble of maintaining a push-down stack to simulate recursion, why not write an augmented push-down machine parser? (See P. M. Lewis II, D. J. Rosenkrantz, and R. E. Stearns, *Compiler Design Theory*, Addison-Wesley, 1976.) Such an algorithm requires maintaining a stack of only the symbols in the same LL(1) grammar used for recursive descent parsing. Logically, the two algorithms are equivalent since each subroutine call in recursive descent corresponds to a symbol in the grammar. By doing "recursive descent" in BASIC, he is saddled with stacking subroutine calls *and local variables* – making his stack larger than necessary when memory is at a premium. In addition, the additional work needed to simulate recursion complicated his code and probably added enormously to the debugging time. A table-driven augmented push-down machine parser would have been shorter and simpler.

While I'm impressed by his having actually gotten Augusta up and running, it seems to be a classic example of an unnecessarily complicated solution to a problem resulting from choosing the wrong design and working with the wrong tools. I wouldn't recommend to others to follow that lead.

Very truly yours,
Clyde B. Schechter
520 West 122 Street
New York, NY 10027

Reply to Clyde Schechter:

Clyde Schechter raises some questions that other readers may also be curious about so I'll try to respond as best I can.

To put the Augusta project in proper perspective: Augusta was begun as a hobbyist project. That means it was done essentially for the sake of doing it, in the sense that rock climbers attempt difficult climbs solely for the sake of the climb.

Obviously, as I explained in Part IV of the Augusta series, neither BASIC nor an Osborne I was the ideal development vehicle for a large software project. But in view of the hobbyist nature of Augusta's development, purchasing a "real" computer was out of the question.

Languages other than BASIC were available for the Osborne I. But due to the 92K disk size on the Osborne I, the other languages were effectively useless. At least one of the languages required almost 140K for the compiler alone. That leaves little room for source code. And since the Augusta compiler had yet to be written, I could only estimate how much disk space I would need for source and object modules.

As I mulled over the choices, Microsoft BASIC seemed to be the best, a small program itself, with both interactive and compiled versions available. In addition, Tiny-Pascal, a project similar in scope to Augusta, had also been written in BASIC, so I knew that it was "do-able." In fact, the bulk of Augusta was written in about five weeks, working only part time. Once compiled with the BASIC compiler, Augusta was compiling at over 300 lines per minute, which compares quite favorably to other microcomputer compilers. So all in all, it really wasn't that difficult to write the compiler in BASIC. (The greatest difficulties were with the p-code interpreter which required several months of writing and tweaking.)

Mr. Schechter correctly observes that it's relatively easy to write a recursive descent parser in a language that permits

EDITORIAL

Here at *DDJ*, as we reach the end of 1983 and look back over the past twelve issues, we are encouraged by what we see. The number of pages is roughly twice what it was a year ago. Circulation has increased substantially. The new faces that have appeared in the columns have brought with them fresh enthusiasm and perspectives. The somewhat altered face of the magazine makes it a bit easier to find on the newsstands, and also a bit more readable. Our group of technical referees has been expanded, and authors are now being paid.

In the midst of all this change, we have fought to maintain our focus. Your responses to our recently completed reader survey have given us additional insight into what we are doing right and what we can improve, what you want and what you are doing. Our thanks to all those who responded.

Also helpful throughout the year has been the Editorial Response insert card. Please continue to use it to vote for your favorite item in the issue or to drop us short comments. The Doctor is never too busy to listen, and the postage is covered.

As we leave 1983, we look for 1984 to be one of the best years yet for *DDJ*. All of us here are rededicating ourselves to the task of improving the *Journal*. Our thanks to all of you for your continuing support. We wish you a happy holiday season, and look forward to serving you in the New Year.

Reynold Wiggins

recursive subroutine calls. Prior to writing Augusta I had worked on both a recursive descent parser for a BASIC interpreter (written in Pascal) and a table-driven LL parser for a subset of Pascal (written in Simula). Based on that experience, I found recursive descent to be easier to work with. In addition, table-driven predictive parsers are more difficult to implement in a one-pass compiler, and a one-pass compiler was an important design goal.

To sum up, his points are well taken. Undoubtedly it could have been done easier with a different set of tools. However, the overall development of the Augusta compiler was not as difficult as some may imagine it to be. Further, my plans all along have been to rewrite Augusta in itself. Having the original compiler as a model simplifies the rewrite, since the new compiler is also based on recursive descent parsing.

As of this writing, most of Augusta has been rewritten in Augusta. Unfortunately, I have been busy with other projects, including writing *Software Building Blocks for the IBM PC* (Hayden, 1984), as coauthor of Software Publishing Corporation's PFS:WRITE, and the teaching of a college summer school class. I will, however, be back working on Augusta in the near future.

I do appreciate everyone's comments, critical or praiseworthy. Thanks for writing.

Edward Mitchell

S-100 to Winchester Improved

Dear Editor,

David Cortesi has suggested some changes to my Winchester interface board (*Dr. Dobb's Journal,* October 1983) which would extend the range of CPU clock frequencies with which it would work. As an added benefit, the use and adjustment of the one-shots are eliminated.

Note that the modified schematic (Figure 1, below) includes a wait state generator, capable of generating one or two wait states, either of which may be selected by appropriately connecting the jumper. Detailed information about wait state generators may be found in Chapter 5 of *Interfacing to S/100-IEEE 696 Microcomputers* by Sol Libes and Mark Garetz, Osborne/McGraw-Hill, 1981. If additional wait states are needed, say for an 8 MHz microprocessor, then the D-type flip-flop could be replaced by a shift register. Details about the use of such a device are to be found in the same book.

Sincerely yours,
Oscar Goldman, Professor
Mathematics Department
University of Pennsylvania
Philadelphia, PA 19104

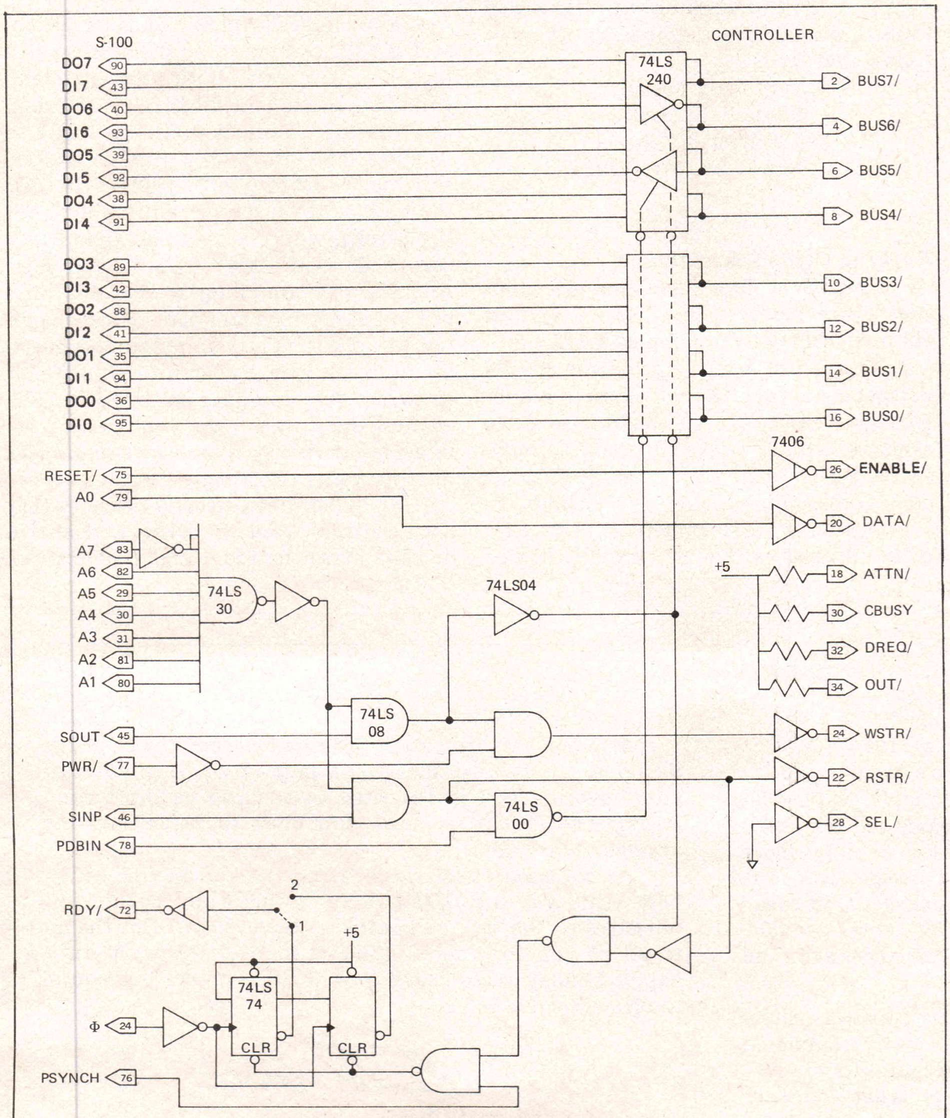


Figure 1.
The improved circuit diagram.

Recovering Lost WordStar Text

Dear *DDJ*,

Every now and then, for various reasons, I lose what I have just typed into WordStar. This induces a case of helpless rage. It's not the retyping I mind, it's the rethinking. I just can't face it.

The last time this happened to me, I didn't give up. WordStar was in an endless loop: beeping and inserting exclamation points in weird and what would otherwise be fascinating patterns. The keyboard was dead. Pulling out the printer interface card while the machine was running (not recommended!) stopped the malignant activity, but the keyboard was still dead. Eventually, there was nothing for it but CTRL-RESET. There went all my work...

But, wait. Maybe it's still in memory! I called up DDT and used the D command to examine all of memory, from the beginning. Sure enough, at hex 7849 on, there were my precious words! I used the DDT M command to move my text to 0100. Back to the system with CTRL-C. Save the recovered material (e.g., SAVE 6 HOPE). Not only can it be typed (TYPE HOPE), but I could use it as input to WordStar with no problems! (Well, there was garbage at the end, a small price to pay, and no doubt inserting a proper terminator with DDT would have cured that.)

Too late to recover that three-page letter I wrote a month ago – but I'm ready for next time. Don't wait until it happens to you. Practice now. Be prepared!

Keith K. Davison
239 Clinton Road
Brookline, MA 02146

More Circles and Such

We received several items amplifying the circle generation discussion begun by Daniel Lee in the May 1983 DDJ. *In order to present them all together, this month the Letters column spills over to the sidebars on pages 14-16. – Ed.*

COHERENT™ IS SUPERIOR TO UNIX* AND IT'S AVAILABLE TODAY ON THE IBM PC.

Mark Williams Company hasn't just taken a mini-computer operating system, like UNIX, and ported it to the PC. We wrote COHERENT ourselves. We were able to bring UNIX capability to the PC with the PC in mind, making it the most efficient personal computer work station available at an unbelievable price.

For the first time you get a multi-user, multitasking operating system on your IBM PC. Because COHERENT is UNIX-compatible, UNIX software will run on the PC under COHERENT.

The software system includes a C-compiler and over 100 utilities, all for $500. Similar environments cost thousands more.

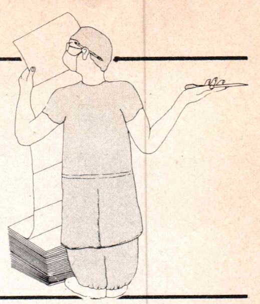
COHERENT on the IBM PC requires a hard disk and 256K memory. It's available on the IBM XT, and Tecmar, Davong and Corvus hard disks.

Available now. For additional information, call or write,

Mark Williams Company
1430 West Wrightwood, Chicago, Illinois 60614
312/472-6659

Mark
Williams
Company

DR. DOBB'S CLINIC



by D. E. Cortesi, Resident Intern

Readers of this column should know about two recent startups in the column industry. The first is a big-time effort that, we think, faces grave difficulties. The second promises to be a gem of great value and deserves to be widely read.

Big-Time Columnizing

Many of you will already know that Douglas Hofstader has retired his "Metamagical Themas" in *Scientific American* and that Brian Hayes has stepped into that hallowed space (originally occupied by Martin Gardner's "Mathematical Games") with "Computer Recreations." His first effort, a consideration of the fundamental shift in thinking encouraged by spreadsheet calculators, appeared in the October issue. It's a welcome sign when the august *SciAm* recognizes the prevalence, and the fun, of personal computing, and we wish Hayes the best of luck.

System-Dependent Prose

However, columnizing isn't easy in the best of circumstances, and he faces some difficulties. We wonder how (given the magazine's huge circulation, the diversity of its readers' interests, and the diversity of the computers they use) he is going to proceed. Spreadsheets are safe enough for a first outing; everybody has one. But when it comes time to present algorithms, what language will he use? When it is time to draw a graph, can he avoid hardware dependency?

The infernal, niggling differences between one language (machine, keyboard, disk drive, etc. *ad nauseum*) are the curse of computer publishing, and nobody has found a decent way to produce system-independent prose. It isn't a new problem, either. The Association for Computing Machinery (ACM) faced it a long time ago and decided to publish algorithms in "presentation ALGOL," the ALGOL syntax augmented with a complicated set of typographical conventions. Donald Knuth faced it and opted to design his own CPU, MIX, and to present algorithms in its assembly language, MIXAL. These solutions are essentially the same. They proceed from the assumptions that (1) no two readers use quite the same language, and (2) each reader knows some language and system very well. Therefore all readers will be presented with a language that *nobody* knows and will all have to learn it; afterward they can translate to their own environment.

The personal-computer press usually operates on the theory that everybody knows and uses some form of BASIC, and that all readers are good enough programmers to convert from one BASIC's idiosyncracies to those of their own machine's BASIC. It isn't true, of course. We once published a program for the IBM PC (in another magazine) and later received a long distance call from an unhappy reader who simply couldn't understand why the program didn't work on his Victor 9000.

Another solution is to embrace the problem and publish system-dependent material only; hence we have magazines for the Apple, the Atari, the Color Computer, the IBM PC; for CP/M; for the Heathkit product line; for machines that use the 6502. And we have books that appear in multiple editions with only trifling changes in their content, like the series of *The Power of XXXXCalc* books. This isn't a solution at all, of course; it only hides the problem. It drives up costs, fragments the audiences, and prevents the dissemination of good solutions to many users.

So how will Hayes handle the problem? Can he find a way to present problems and solutions to a mass audience in a way that will be interesting and useful to most of his readers? We'll see.

Bently's Pearls

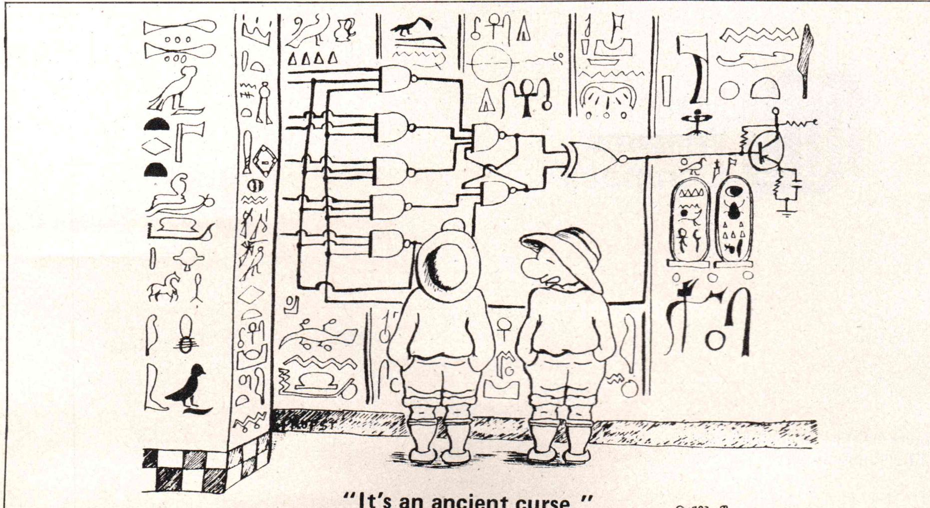
Communications of the ACM, longtime flagship of computer science journals, launched a new column in its August issue. The columnist is Jon Bently, an Associate Professor at Carnegie-Mellon and author of *Writing Efficient Programs* (a book that, on the evidence of Bently's debut column, we certainly plan to buy).

The column's title is "Programming Pearls," the pearls being those parts of programs whose origination require "insight and craftsmanship." The pearls presented in the first column were delightful and gave us several hours of vigorous, enjoyable mental exercise. Bently has avoided the problem of system dependency by keeping his problems abstract and by presenting algorithms in English and, where necessary, in a simple pseudo-code.

Bently plans to end each column with questions and problems, but he isn't asking for solutions from his readers (from our experience, we'd call that a mistake; the massed mental power of a motivated readership is awesome). Here is one of the problems he presented.

"Rotate a vector of N elements by I positions. For instance, with $N=8$ and $I=3$, the vector *ABCDEFGH* is rotated to be *DEFGHABC*. It is easy to do this rotation in N steps if we have available an N-element intermediate vector. Can you rotate the vector in time proportional to N using only a few extra words of storage?"

We could; we ended up using five scratch variables. Furthermore, when we expressed our solution as a BASIC subroutine (taking variables I and N and vector V as input, modifying the contents of V), we found that with a single added line of code we could accept a negative value of I to mean "rotate right." Can you find a similar pearl for this oyster of Bently's? If you enjoy searching for that one, be sure to read the rest of his column. DDJ



"It's an ancient curse."

by Anthony Skjellum

In the last column, several references to the book *The C Programming Language* and its authors were made. Through my error, Brian Kernighan's name was misspelled consistently throughout the column. I'm sure that many readers noticed this immediately. Unfortunately, I didn't until I saw the column in print. [*We obviously missed it too. Our apologies. –Ed.*]

In this second Programmer's Notebook, I'll discuss how Unix-type environments can lead to non-interactive, and user-unfriendly, software. This is based on experiences I've had with several Unix and Unix-like systems running both standard and Berkeley Unix.

Skimming through the ads of the October *DDJ,* I noticed a very interesting item: the Computer Innovations ad concerning the soon-to-be-released version of their C compiler. As an option, this compiler will produce programs/data exceeding the previous limit of 64K segments imposed by all 8086 compilers (C or otherwise) that I've seen. Look for a review in this column early next year.

1. Terse (hard-to-remember) program names.
2. Lack of program sign-on and sign-off messages.
3. Lack of interactive mode to alleviate the need to re-execute a program several times to complete a set of operations.
4. Inconsistent use of switch (dash options) for controlling the specifics of program execution.
5. Lack of descriptive error messages.
6. Cryptic, incomplete, and erroneous documentation.
7. Software bugs: undocumented and documented.
8. Cryptic (or missing) internal help features.
9. Poor console interface provided by Unix.
10. Lack of system for finding program names by the function required.

Table 1
Unix Software Problem Areas

```
$ fct <RETURN>                    (activate the program with no arguments)
usage: fct -abcv file1 . . fileN  (help message)
$                                 (shell prompt)
```

Figure 1.
Cryptic fct Session

```
$ fct <RETURN>              (activate the program with no arguments)
fct (version x.yz) as of dd-mm-yy
usage:
        fct [dash options] file1 . . fileN
dash options:
        -a      perform function "a" on files specified
        -b      perform function "b" on files specified
        -c      perform function "c" on files specified
        -v      verify each step before proceeding
note:   -a and -b are mutually exclusive; -c may be used in conjunction
        with -b only.
End of execution.
$                           (shell prompt)
```

Figure 2.
Friendly fct Session

Unix Software

The Unix operating system was designed to reduce repetition of programming effort by permitting modular programs to be combined via pipes and tees. Since input and output are redirectable under Unix, simple programs could use console input and output for one application and be used as part of a pipeline for another. Thus, unmodified programs could be reharnessed for new applications to an extent not possible with previous operating systems.

Pipes and input-output redirection are two of the best and most well-known features of the Unix system. Microcomputer users have been very interested in adding these capabilities to their own operating environments. In the 8-bit world, this has been done chiefly through special subroutine libraries such as "The Unica," or in C runtime packages. For MS-DOS 2.0 users, the features are built into the operating system.

Despite the undisputed usefulness of pipelines and input-output redirection, their presence in Unix has lead to a serious drawback in the system's environment. This drawback is the proclivity to avoid interactive programs and to produce user-unfriendly software. Furthermore, the standard Unix console interface is weaker than under other operating systems. In the remainder of this column I will discuss

these weaknesses as I perceive them.

Non-Interactive Software

Because of the availability of pipes and input-output redirection, many Unix programs are designed to act as filters. Filters are programs which require a single sequential input data stream and produce a single output data stream. Such programs are suitable as pipe-fittings. Because of the way they handle data, they don't normally expect to be used interactively. In most cases this doesn't pose a problem for users. However, because such programs do not expect to deal directly with humans, but only with input and output streams, they can often be very unfriendly in handling errors.

The problem in the Unix operating system is that the same terse philosophy applied to filters also pervades most of the software available. This includes programs which are normally executed sequentially by the user from the console. The problems come in several areas and some of these problem areas are listed in Table I (page 14). The following paragraphs will elaborate each of the points listed in that table.

Terse Names

Unix program names are usually two or three letters long and tend to be cryptic. While this saves typing for experienced users, it's frustrating for new and occasional system users. Also, since the Unix system lacks an on-line indexing system for finding program names by function, it's not easy to find the right program based on the desired function alone.

Sign-ons and Sign-offs

Sign-on and sign-off messages are a common courtesy in the computer world. Virtually all standard Unix programs lack these two simple features. While this is understandable for filters, it is completely unnecessary for other programs. For example, if two versions of a program exist on a system, the only easy way to distinguish them is by their sign-on messages.

Besides sign-ons/sign-offs, it is also nice for a program to give progress reports during execution. This lets the user know how things are proceeding. Standard Unix software doesn't normally include such a feature.

Internal Help Features

Many programs include a feature summary option to help occasional and new users remind themselves of program operation. Many Unix programs also have this capability, but they are often extremely cryptic and include few English words to supplement the sample command line which they display. Figure 1 (page 14) displays a sample session in which a fictitious program, **fct**, is executed from the shell with no arguments. The program responds with a cryptic help summary typical of actual Unix commands. In Figure 2 (page 14) the same **fct** program session is presented, but this time the program has been designed to provide a user-friendly help feature (and also to sign on and off).

Interactive Modes

Interactive program modes provide a friendly environment for the user. When a program is used often, it may be executed several times consecutively. An interactive mode eliminates the need for consecutive execution since the user can enter all the commands in one interactive session. This avoids unnecessary user effort and is probably more efficient from a system standpoint.

One reason that the interactive modes are missing is the lack of support for expanding Unix wildcard filenames from within a program. I find this limitation rather arbitrary, and I have written a program which solves the problem (see "Expanding Unix Wildcards," *DDJ*, No. 73, November 1982). The lack of direct support for such a function indicates that the whole Unix philosophy is geared towards non-interactive software tools.

Command-line Switches

Unix programs use command-line switches (dash options) to specify the particulars of program execution. Switches usually consist of a dash character ("-") followed by a single letter. Unfortunately, Unix programs use these switches inconsistently. For example, the **rm** command allows only limited positioning of switches, permitting them only before filenames on the command line. Some programs allow several switches to be combined after a single dash character (e.g., -abcdef). Others use the plus ("+") character to activate a program feature and a dash to deactivate it. Generally, it is difficult to remember all the different possibilities, limitations, and defaults imposed by the various programs.

The difficulties with command-line switches under Unix led me to write the ARGUM package which was published in the August 1982 issue of *DDJ* (No. 70). That article proposed a program which would handle switches in a consistent way. Existing Unix programs could be changed to use ARGUM, thus eliminating one degree of inconsistency from the operating environment. Alternatively, the less powerful Unix III **getopts()** facility could be used.

Lack of Descriptive Error Messages

The lack of descriptive error messages is a real problem, especially for inexperienced users. One offender is the **eqn/troff** system used for equation and text phototypesetting. These programs report errors as "Syntax Error" between two line numbers. They don't echo the erroneous text or equation, and the line numbers aren't always useful because header files change the length of the source text.

Another problem stems from the way Unix reports failures. Programs which attempt to open a file and fail get an error code. They subsequently report the failure as "cannot open file." While this is correct, it doesn't tell the user if the file is nonexistent or if a file protection violation has occurred. Similarly, when users try to change their current directory to one for which they have no read privileges, a "bad directory" message is displayed by the Unix shell.

Documentation

Documentation is a real problem in the Unix system. Most programs are documented in a standard form which is typically very terse. The examples provided are often very complicated and don't clearly illustrate how to use the program for simple purposes. Because of the poor documentation, some casual users think of Unix systems as secret societies since only the indoctrinated can tell them how the system and programs work.

Another problem with documentation is that it sometimes doesn't reflect the current state of a program. Most notable on the system I use are undocumented features of the standard Unix editor **ed**. The standard form provides no line edit command ("x") while the installed version does. Yet, the documentation does not explain this.

Unix documentation is clearly the weak link in the system. It makes some of the mysterious concepts of the system seem impossible to grasp and reduces productivity through its terseness.

Software Bugs

Another problem which makes a Unix system user-unfriendly (actually user-hostile) is the presence of undocumented bugs in important software packages. For example, bugs exist in the **eqn/troff** system. These can be circumvented, but the methods are known only to a few experts. No generally available documentation exists for avoiding such problems.

Since the Unix system comes with source code, it should be feasible for individual users to change system software to their specific needs. Unfortunately, the source code which accompanies Unix is mostly comment-free and is therefore difficult to understand without significant effort.

User Console Interface

The user console interface is not extremely good under Unix. For example, when a character is deleted, Unix does not actually remove the character typed but just moves the cursor back one space. There is no standard mechanism for having a line retyped, and backspacing after typing a tab doesn't produce the correct results. Furthermore, when a control character is typed, it is displayed in the form ^CHAR, but when deleted, the cursor moves back only one space (leaving the caret).

The weak user interface points again to the philosophy that most software will not be interactive. However, the standard Unix editor makes no attempt to improve the interface for the purpose of interactive editing. It is so unfriendly that most users resort to other editors (usually screen editors).

Programmers can only provide their programs with a superior console interface by using the raw terminal mode. Unfortunately, this mode is more expensive in terms of input-output cost. For user-friendly, screen-oriented software, it's the only way to go.

Conclusion

Unix is a powerful operating system, and is certainly one that I enjoy using. It does, however, have a number of user-unfriendly aspects, and the system philosophy has led to a predilection for non-interactive software.

I look forward to hearing how other Unix users perceive the Unix environment.

DDJ

Reader Ballot
Vote for your favorite feature/article
Circle Reader Service **No. 193.**

Faster Circles for Apples

Daniel Lee's article, "Fast Circle Routine," in *DDJ* No. 79 (May 1983) inspired me to create a similar circle-making approach for the Apple. Although I first wrote an Applesoft BASIC program to test the logic, I chose variable names that could serve for both BASIC and assembly language. Thus "X" and "Y" refer to the X and Y registers of the Apple's 6502 chip, and "Xcrd" and "Ycrd" are the coordinates for the points of the circle.

A circle of radius R centered at (Xmid, Ymid) is described by the familiar formula

by Myron L. Pulier

Myron L. Pulier, M.D., 101 Cedar Lane, Teaneck, New Jersey 07666.

```
(Ycrd-Ymid)^2
    + (Xcrd-Xmid)^2 = R^2
```

where Xcrd is the horizontal variable, Ycrd is the vertical variable, and Xmid and Ymid are constants. Differentiating with respect to Xcrd gives

```
2*(Ycrd-Ymid)*dYcrd/dXcrd
    + 2*(Xcrd-Xmid) = 0
```

whence

```
dYcrd/dXcrd =
    - (Xmid-Xcrd)/(Ymid-Ycrd)
```

The last equation implies that, in drawing the circle, if we increase Xcrd by 1 to plot the next point we must decrease Ycrd by

```
(Xmid-Xcrd)/(Ymid-Ycrd)
```

The slowest operation here is division by Ymid-Ycrd, which must be performed each time we want a new value for Ycrd. We can reduce the number of these divisions by evaluating the expression for only one eighth of the circle and by plotting the rest of the circle symmetrically about the coordinate axes and about a diagonal.

It is best to select the upper left extreme of the circle as the starting point. According to the Apple coordinate system, where point (0,0) is the upper left corner of the screen, our starting point is given by

```
(Xmid-R/SQR(2),Ymid-R/SQR(2))
```

From here we move to the right and stop at the extreme top of the circle, which is point (Xmid,Ymid-R). This choice of starting and ending points facilitates a simple FOR-NEXT program loop (FOR Xcrd = Xmid-R/SQR(2) to Xmid) and avoids the divide-by-zero error we might encounter at the extreme right and left of the circle, where the slope is undefined.

Reader Commentary

More Fast Circles . . .

Dear *DDJ,*

Daniel L. Lee's algorithm has got to be faster than Microsoft's pedestrian CIRCLE command, but both suffer from the same malady: they reinvent the wheel – only this time it's square!

When I think I've discovered a marvelous algorithm, I wonder if I've outsmarted the professionals. I usually haven't. But hope springs eternal. I search the literature anyway. My brainchild is at least 17 years old [B. K. P. Horn, "Circle Generators for Display Devices," *Computer Graphics and Image Processing* (5), pp. 280-288 (1976)].

Neither trigonometry nor calculus is needed to devise a circle generator. For a circle of radius R one wants to plot points (X,Y) with integer coordinates which most nearly solve the equation

$$X^2 + Y^2 = R^2$$

The difference between the left side and R^2 is a measure of nearness. A suitable circle generator simply chooses successive points to minimize this difference. The enclosed listing (see Listing Three, page 30) is a rendering of such an algorithm. It generates points for about one eighth of the circle and, using the symmetry of the circle, plots eight points for each point generated. For use with digital plotters, the algorithm is invoked eight times forward and backward so that the points are of concentric circles; the low algorithm is plotted in counterclockwise order. I enclose a plot of concentric circles in low resolution to exhibit the algorithm's behavior (see Figure 2, below).

William A. McWorter, Jr.
Mathematics Department
Ohio State University
231 W. 18th Avenue
Columbus, OH 43210

(Listing Three begins on page 30)

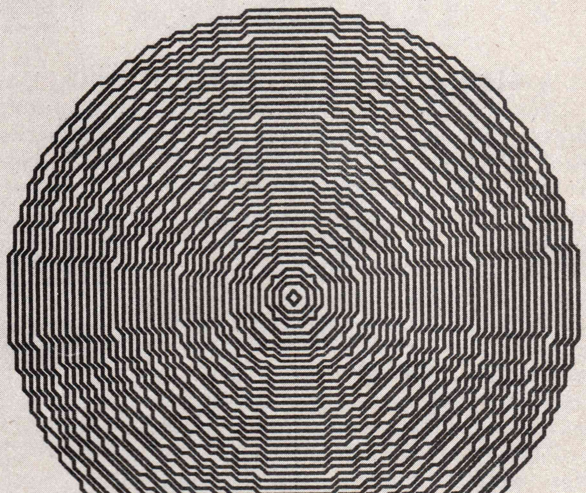


Figure 2.

See Figure 1 (at right).

For each point that we find by the above method, we can generate seven symmetric points by reflection through the horizontal and vertical diameters of the circle and by exchanging Xcrd with Ycrd. A point (Xcrd,Ycrd) is ABS(Xcrd–Xmid) distant from the vertical axis of the circle. Since its reflection should be the same distance from this axis, its abscissa is 2*Xmid–Xcrd. Similarly, reflection through the horizontal diameter gives an ordinate of 2*Ymid–Ycrd. To reflect a point through the diagonal line that runs from upper left to lower right, we find the point whose distance to the vertical axis equals the distance of the original point to the horizontal axis and whose distance to the horizontal axis is the same as the original point's distance from the vertical axis, namely

(Xmid+Ymid–Ycrd,Xmid+Ymid–Xcrd)

This new point can now be reflected as before through the vertical and horizontal axes.

In Listing One (page 21), subroutine 9000 plots four points symmetrically about the horizontal and vertical axes. Lines 10050 and 10055 switch the X and Y coordinates, then line 10060 calls 9000 to plot the four new points. The parame-

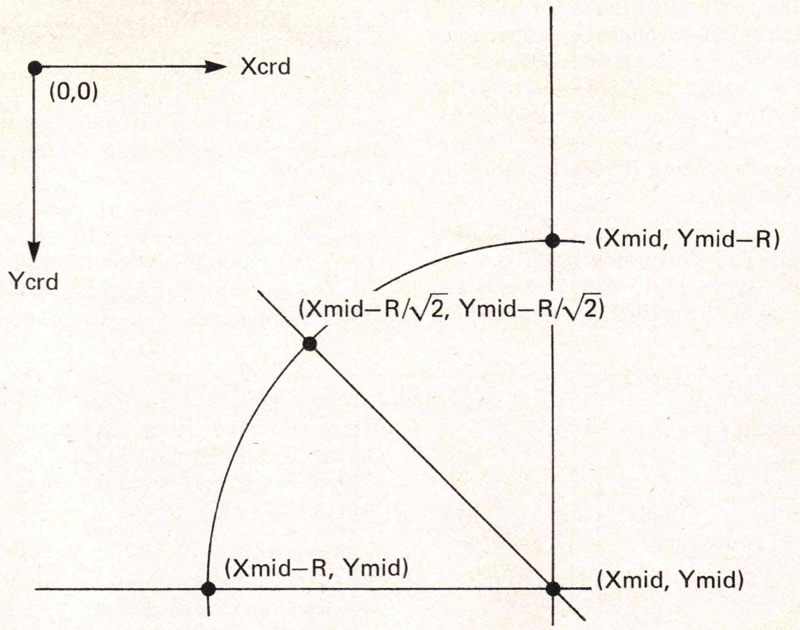


Figure 1.
Faster Circle

Reader Commentary

. . . And Fast Ellipses

Dear Sirs:

Mr. Daniel Lee, in the May '83 issue, presented a fast circle generator. It compensated for any given screen aspect ratio, and as such may be used as an ellipse generator. I submit the algorithm described below as an even faster alternative. The speed improvement results from the elimination of all division and most of the multiplication. The approach taken could easily be modified to allow the generation of arcs.

The method which I present here is based on the equation of the circle, and a trick which eliminates a great deal of multiplication. There is no calculus or trigonometry involved, implicitly or explictly.

The equation of the circle is well known:

$$x^2 + y^2 = r^2 \qquad [1]$$

where r is the radius. Since we want to minimize multiplication, we have to use "magic." A magical property of the positive integers is that the square of a positive integer *n* is the sum of the first *n* odd numbers. This means that if we want to compute x^2 for each x we can actually plot (i.e., each integer x), we only need to know which odd numbers to add up. The same applies to y^2.

In order to plot a circle, we might start at the point (0,r) and plot towards (r,0), using symmetry to generate the other arcs of the circle. This would mean that x would go from 0 to r, y would go from r to 0, x^2 would go from 0 to r^2, and y^2 would go from r^2 to 0. It is easier than it first appears to calculate y^2. Note that y^2 is the sum of the odd numbers from 1 to $2y-1$. In the initialization phase it will be necessary (perhaps) to compute y^2 directly, but for $y' = y-1$, $y'^2 = y^2 - (2y-1)$.

Above I said "perhaps" because it develops that one does not need to refer directly to y^2 or even to x^2. The procedure for drawing the circle requires that we assume, as we did above, that we will draw primarily from (r,0) to (0,r) and use symmetry to generate the rest of the points. As we compute the points for the primary arc, we maintain a total *e*. The total starts at 0; for every time we actually move in the positive x direction, we add $2x-1$ to *e*; for every time we actually move in the negative y direction, we subtract $2y-1$ from *e*. We decide precisely which step or combination of steps to take by insisting that the *e* that would result from the step or combination of steps be as close to 0 as possible.

An Ellipse

To generate an ellipse is a slightly more complex matter, but in the end we lose little speed. The equation for an ellipse centered at the origin is

$$b^2x^2 + a^2y^2 = a^2b^2 \qquad [2]$$

where b is the positive y-intercept, a is the positive x-intercept, and a/b is the resulting aspect ratio. I claim that in order to successfully trace the ellipse we need only do exactly as we do for the circle, but we must multiply every reference to x by b^2 and every reference to y by a^2. In other words, every time we actually move in the positive x direction, we add $b^2(2x-1)$ to *e*; for every time we actually move in the negative y direction, we must subtract $a^2(2y-1)$ from *e*. Again we decide which step or combination of steps to take by insisting that the *e* that would result from the step or combination of steps be as close to 0 as possible. In this case we are plotting from (0,b) to (a,0).

If perhaps the terms $b^2(2x-1)$ and $a^2(2y-1)$ look like they involve too much multiplication, please realize that in fact no multiplication is required. For example, we would already know the evaluation of $b^2(2x-1)$ to

(Continued in box on page 20)

ters R2, X2, Y2, and XY have been introduced to speed computation.

In the segment of the circle from the upper left point through the upper middle, the change in Ycrd is fractional for each unit change in Xcrd. Because the Apple plotting routine deals with integers, the decrement in Ycrd builds until it causes the line being drawn to move up one full position.

The assembly program in Listing Two (page 22) runs much faster than its Applesoft equivalent. Since Xcrd can range from 0 through 279, it must be a double-precision variable. It occupies locations XCRDH and XCRDL. Ycrd is supplemented by a fractional portion stored in YCRDF. Names of other double-precision integer parameters are terminated with -H or -L for the high- and low-order portions, respectively. Single-precision assignment is indicated in the comments by "<—", while double precision is "<<—".

The TEST program plots a circle of radius 40 and midpoint (120,80). It initializes the hires screen by calling TURNON. The subroutine called EIGHTH performs calculations for the one-eighth circle. Here the first order of business is to approximate the value of R/SQR(2) by using R*3/4 instead. Note that 3/4 in decimal is 1/2 + 1/4, or 0.11 in binary.

The next lines of the assembly program are a straightforward translation of their Applesoft equivalents. Lines 75 and 76 initialize the value of YCRDF to 0. PLOTFOUR is called in lines 104 and 119 to place four points symmetrically about the horizontal and vertical axes of the circle. PLOTFOUR uses the Applesoft HPLOT routine to perform the actual plotting. HPLOT requires that the horizontal coordinate be in the Y and X registers,

(Continued from page 19)

be, say, e_x. To determine $e_{x'}$ when $x' = x+1$, note that

$$b^2(2x'-1) = b^2[2(x+1)-1] = b^2(2x-1) + 2b^2;$$

in other words,

$$e_{x'} = e_x + 2b^2.$$

A similar result obtains for the negative y direction, which we will simply state:

$$e_{y'} = e_y - 2a^2.$$

Algorithm Summary

To summarize the algorithm: start with the point (0,b). Initialize e to 0, e_x to b^2, e_y to $2a^2b-a^2$, e_{xy} to e_x+e_y. Plot the current point and corresponding points in the other quadrants of the ellipse. Choose the next point so that e plus e_{whatever} is minimized. Set e according to that choice, and update e_x, e_y, and e_{xy}. When the point (a,0) is arrived at, the ellipse is complete.

The Listing

The program shown in Listing Four (page 30) is an MBASIC program intended to interface to an LSI ADM-3A terminal. Obviously, if speed is a concern, BASIC is not the language of choice. I chose it to permit the program to be tried out basically anywhere, since my facilities for computer graphics are one-of-a-kind.

Lines 1050-1240 are the routine itself. The point-plotting routine is on lines 1310-1341.

Caveat

There is one thing that the implementor should be aware of before he or she starts, to prevent untraceable bugs. The formulae for e_x, e_y, and e_{xy} include squares of a and b. These squares accumulate to a large total rather quickly. The solution is to use a wide word to store the total, and perhaps (depending on the size of your screen in pixels) the values of e_x, e_y, and e_{xy} as well.

Drawing Arcs

The method can be modified to draw arcs (see Figure 3, below) elliptical or otherwise, with careful initialization and a well-considered termination condition. The initialization involves calculating e_x, e_y, and e_{xy} for the initial point of the arc to be drawn. The routine should terminate when the last point of the arc is drawn. The actual coordinates of the final point should be calculated in some fashion that allows for rational numbers, and then a point with integer coordinates should be chosen that approximates the actual point. This can be done by using the equation of the ellipse. In other words, the best integer approximation (x_i,y_i) of the terminating point (x,y) is the one for which $(bx_i)^2+(ay_i)^2$ is closest to $(ab)^2$. Again, the integer coordinates of the final point should be computed in the initialization phase and used as the termination condition.

Conclusion

This routine can draw an ellipse quickly, using no multiplication once initialized. It should be easily implemented in 68000 assembly language, owing to that processor's 32-bit register operations. A little more difficulty should be anticipated by users of the 8086, 6809 or Z80, though their 16-bit addition capabilities can be used to advantage. HLLs can speedily draw circles with this routine, as well, because of its incremental nature. And finally, the algorithm can draw arcs easily.

Michael T. Enright
2360 Hosp Way, #132
Carlsbad, CA 92008

(Listing Four begins on page 30)

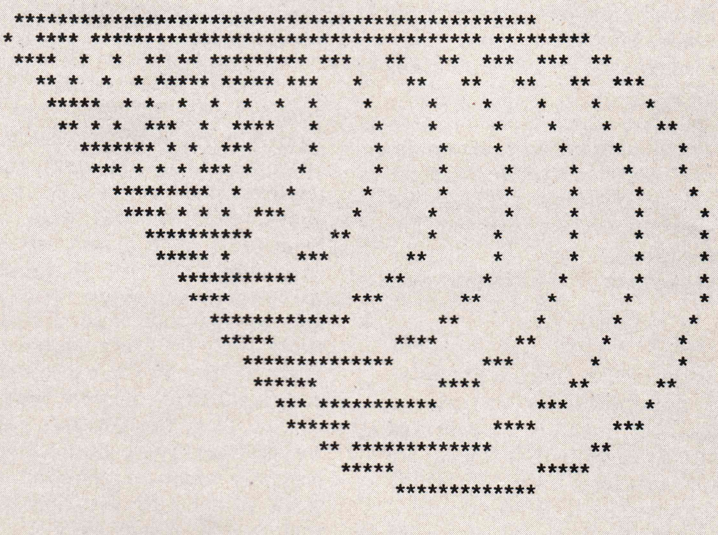


Figure 3.

and the vertical coordinate in the A register. HPLOT returns these coordinate values at the zero-page locations named LASTHH, LASTHL, and LASTV.

The division sequence starts on line 135. Since the divisor and dividend are single precision, we can use a technique that divides a one-byte divisor stored in DIVISOR into a one-byte dividend held in the A register. The eight shift operations and eight subtractions required are counted via the X register. The result of the division is a binary fraction generated in QUOTIENT. This quotient is subtracted from the previous value of YCRDF. If a borrow is required, we decrement the integer portion of Ycrd. In any case, Xcrd must be incremented by 1 in a double-precision operation. The FOR-NEXT loop of the BASIC version is implemented in assembly language by counting the value of R2 down through zero, since R/SQR(2) points will be plotted for one eighth of the circle.

Using zero-page locations for variables and parameters and a faster division algorithm will increase speed, but the bottleneck is the Applesoft HPLOT routine, which maps horizontal and vertical coordinates into the Apple video locations. Replacing that routine with a table lookup results in very fast circle generation.

With some modification the "faster circle" technique can produce filled-in disks or wedges for pie charts. It can also rotate and translate shapes and objects quickly for animation effects.

(Listings begin below)

Circles (Text begins on page 18)

Listing One

```
1000    ***************TURNON

1005    Hgr
1010    Hcolor = 3


2000    ***************TEST

2005    R = 40
2010    XMID = 120
2015    YMID = 80
2020    Gosub 10000
2025    End


9000    ***************PLOTFOUR

9005    Hplot H,V
9010    Hplot X2 - H,V
9015    Hplot X2 - H,Y2 - V
9020    Hplot H,Y2 - V
9025    Return


10000   **************EIGHTH

10005   R2 = R * .7
10010   YCRD = YMID - R2
10015   X2 = XMID + XMID
10020   Y2 = YMID + YMID
10025   XY = XMID + YMID
10030   For XCRD = XMID - R2 To XMID
10035           H = XCRD
10040           V = YCRD
10045           Gosub 9000
                        *   PLOT 4 POINTS
10050           V = XY - XCRD
10055           H = XY - YCRD
10060           Gosub 9000
                        *   PLOT 4 POINTS

10065           YCRD = YCRD -
                        (XMID - XCRD) /
                                (YMID - YCRD)

10070   Next
10075   Return

END OF LISTING

PROGRAM LENGTH = 421 BYTES,
        TOTAL OF 31 LINE NUMBERS


27 TOTAL NON-REM STATEMENTS,
        6 TOTAL REMARKS


END
PR#0
```

End Listing One

(Listing Two begins on page 22)

Circles (Listing continued, text begins on page 18)

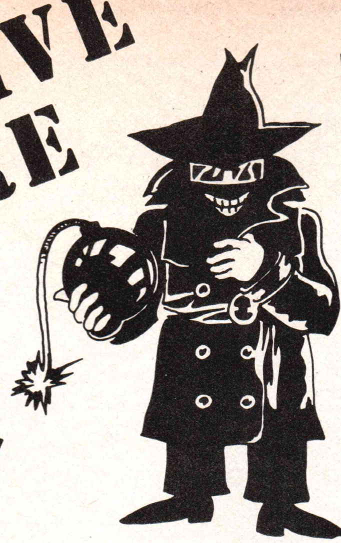
Listing Two

```
0800                     LST
0800 2C 50 C0  TURNON    BIT GRAPHICS
0803 2C 52 C0            BIT FULL
0806 2C 54 C0            BIT PAGE1
0809 2C 57 C0            BIT HIRES
080C A9 FF               LDA #$FF
080E 85 E4               STA HCOLOR
0810 A9 20               LDA #$20
0812 85 E6               STA HPAGE
0814           ;
0814 A9 28     TEST      LDA #40             ;R<-40
0816 8D 2F 09            STA R
0819           ;
0819 A9 78               LDA #120            ;XMID<<-120
081B 8D 37 09            STA XMIDL
081E A9 00               LDA #0
0820 8D 36 09            STA XMIDH
0823           ;
0823 A9 50               LDA #80             ;YMID<-80
0825 8D 3C 09            STA YMID
0828           ;
0828 20 64 08            JSR EIGHTH          ;DRAW CIRCLE
082B 60                  RTS
082C           ;
082C AC 2D 09  PLOTFOUR  LDY HH              ;HPLOT H,V
082F AE 2E 09            LDX HL
0832 AD 31 09            LDA V
0835 20 57 F4            JSR HPLOT
0838 AD 33 09            LDA X2L             ;HPLOT X2-H,V
083B 38                  SEC
083C E5 E0               SBC LASTHL
083E AA                  TAX
083F AD 32 09            LDA X2H
0842 E5 E1               SBC LASTHH
0844 A8                  TAY
0845 AD 31 09            LDA V
0848 20 57 F4            JSR HPLOT
084B A4 E1               LDY LASTHH          ;HPLOT H2-H,Y2-V
084D A6 E0               LDX LASTHL
084F AD 3B 09            LDA Y2L
0852 38                  SEC
0853 E5 E2               SBC LASTV
0855 20 57 F4            JSR HPLOT
0858 AC 2D 09            LDY HH
085B AE 2E 09            LDX HL
085E A5 E2               LDA LASTV
0860 20 57 F4            JSR HPLOT
0863 60                  RTS
0864           ;
0864 AD 2F 09  EIGHTH    LDA R               ;R2<-R*3/4
0867 4A                  LSR
0868 18                  CLC
0869 6D 2F 09            ADC R
```

(Continued on page 24)

SUBVERSIVE SOFTWARE

So cheap and useful it's...dangerous!



A radical idea in software development.
Useful Pascal programs, with:

- Complete annotated *source listings;*
- Disk(s) with *source code and compiled code;*
- User manual;
- Complete programmer documentation describing data structures and algorithms; and giving suggestions for modification.

Modify them, include them in your own systems, or simply use them.

A growing library of software tools you'll find hard to resist.

SUBVERSIVE SOFTWARE

TPL

The Text Processing Language. A text-file runoff program consisting of a set of text-processing primitive commands from which more complex commands (macros) can be built (as in Logo). Features include:

- Complete customization of text processing through macro definition and expansion, looping structures, and conditional statements;
- Adapts to any printer;
- Pagination;
- Text justification and centering;
- Indexing and tables of contents;
- Superscripts and subscripts;
- Bolding and underlining;
- Multiple headers and footers;
- End notes and footnotes;
- Widow and orphan suppression;
- Floating tables and 'keeps.'

$50

SUBVERSIVE SOFTWARE

DBX

Blocked Keyed Data Access Module. Maintains disk files of keyed data. Can be used for bibliographies, glossaries, multi-key data base construction, and many other applications.

- Variable-length keys;
- Variable-length data;
- Sequential access and rapid keyed access;
- Single disk access per operation (store, find, delete) in most cases;
- Multiple files;
- Dynamic memory allocation for RAM-resident index and current "page" of entries;
- Includes demonstration program and testbed program.

$50

SUBVERSIVE SOFTWARE

PDMS

The Pascal Data Management System. A user-oriented data management system in which numeric and alphanumeric data are stored in tables with named columns and numbered rows. Currently being used for dozens of different kinds of business and scientific applications, from inventory management to laboratory data analysis. Includes over 20 Pascal programs; more than 10,000 lines of code. Main features include:

- Maximum of 32,767 rows per file;
- Maximum of 400 characters per row, and 40 columns per table;
- Full-screen editing of rows and columns, with scrolling, windowing, global search/replace, and other editing features;
- Sorting, copying, merging, and reducing routines;
- Mailing label program;
- Reporting program generates reports with control breaks, totals and subtotals, and selects rows by field value; many other reporting features;
- Cross-tabulation, correlations, and multiple regression;
- Video-display-handling module;
- Disk-file-handling module.

Many other features. UCSD formats only.

$250

SUBVERSIVE SOFTWARE

ZED

Full-screen text editor; designed to be used either with TPL or by itself.

- Full cursor control;
- Insert mode with word wrap;
- 'Paint' mode;
- Single-keystroke or dual-keystroke commands;
- Command synonyms;
- Global search and replace;
- Block move, block copy, and block delete.

$50

BVERSIVE FTWARE

SCINTILLA

A log logit curve fitting program for radio-immunologic data; must be used with PDMS (described above).

- Multiple protocol files;
- Quality control files;
- Four-parameter non-linear curve fit.

UCSD formats only. **$250**

SUBVERSIVE SOFTWARE

CHROME

Chromatography data analysis program:

- Graphic display of analog data;
- Panning and zooming;
- Automatic peak-finding and baseline calculation;
- Full interactive peak editing;
- Computation of peak areas;
- Strip charts on C. Itoh and EPSON printers.

$100

SUBVERSIVE SOFTWARE

PLANE

Planimetry program:

- Bit-pad entry of cross sections;
- Real-time turtlegraphics display;
- Calculation of areas;
- Saves calculations to text file.

$100

VERSIVE TWARE

MINT

A terminal emulation program for communication between computers of any size.

- User-configurable uploading and downloading of files;
- X-ON/X-OFF and EOB/ACK protocols;
- Interrupt-driven serial input (for Prometheus Versacard in Apple II);
- Printer-logging.

$50

For more information, call 919-942-1411. To order, use form below or call our toll-free number: 1-800-XPASCAL

Check appropriate boxes:

FORMAT	PRODUCT	PRICE
☐ 8″ UCSD SSSD	☐ DBX	$ 50
☐ 5¼″ Apple Pascal	☐ PDMS	$250
☐ 5¼″ UCSD IBM PC 320k	☐ TPL	$ 50
☐ 8″ CP/M SSSD	☐ ZED	$ 50
☐ 5¼″ IBM MS-DOS	☐ MINT	$ 50
☐ 5¼″ CP/M Osborne	☐ SCINTILLA	$250
	☐ CHROME	$100
	☐ PLANE	$100

Name ______________________________

Address ______________________________

☐ MasterCard ☐ VISA ☐ Check ☐ C.O.D.

(Please include card # and expiration date)

SUBVERSIVE SOFTWARE A division of Pascal & Associates,
135 East Rosemary St., Chapel Hill, NC 27514

Circle no. 76 on reader service card.

Circles (Listing continued, text begins on page 18)

Listing Two

```
086C 6A                 ROR
086D 8D 30 09           STA R2
0870           ;
0870 AD 3C 09           LDA YMID         ;YCRD<-[YMID-R2]
0873 AA                 TAX              ;X<-YMID FOR LATER
0874 38                 SEC
0875 ED 30 09           SBC R2
0878 8D 3D 09           STA YCRD
087B           ;
087B AD 37 09           LDA XMIDL        ;X2<<-[2*XMID]
087E 0A                 ASL
087F 8D 33 09           STA X2L
0882 AD 36 09           LDA XMIDH
0885 2A                 ROL
0886 8D 32 09           STA X2H
0889           ;
0889 8A                 TXA              ;A<-YMID
088A 0A                 ASL              ;Y2<<-[2*YMID]
088B 8D 3B 09           STA Y2L
088E A9 00              LDA #0
0890 8D 3E 09           STA YCRDF        ;YCRDF<-0
0893 2A                 ROL
0894 8D 3A 09           STA Y2H
0897           ;
0897 8A                 TXA              ;A<-YMID
0898 18                 CLC              ;XY<<-[XMID+YMID]
0899 6D 37 09           ADC XMIDL
089C 8D 39 09           STA XYL
089F A9 00              LDA #0
08A1 6D 36 09           ADC XMIDH
08A4 8D 38 09           STA XYH
08A7           ;
08A7 AD 37 09           LDA XMIDL        ;XCRD<<-[XMID-R2]
08AA 38                 SEC
08AB ED 30 09           SBC R2
08AE 8D 35 09           STA XCRDL
08B1 AD 36 09           LDA XMIDH
08B4 E9 00              SBC #0
08B6 8D 34 09           STA XCRDH
08B9           ;
08B9 AD 35 09  NXPOINT  LDA XCRDL        ;H<<-XCRD
08BC 8D 2E 09           STA HL
08BF AD 34 09           LDA XCRDH
08C2 8D 2D 09           STA HH
08C5           ;
08C5 AD 3D 09           LDA YCRD         ;V<<-YCRD
08C8 8D 31 09           STA V
08CB           ;
08CB 20 2C 08           JSR PLOTFOUR     ;PLOT SET OF POINTS
08CE           ;
08CE AD 39 09           LDA XYL          ;H<<-[XY-YCRD]
08D1 38                 SEC
08D2 ED 3D 09           SBC YCRD
```

(Continued on page 26)

Z80 Software

DEBUG · QED · LINKER · MODEM · MACRO · TPM · BASIC I · ZDDT · ZTEL · QSAL · ZEDIT · TOP I · 6502X · TPM II

SOFTWARE DESCRIPTIONS

TPM (TPM I) - $80 A Z80 only operating system which is capable of running CP/M programs. Includes many features not found in CP/M such as independent disk directory partitioning for up to 255 user partitions, space, time and version commands, date and time, create FCB, chain program, direct disk I/O, abbreviated commands and more! Available for North Star (either single or double density), TRS—80 Model I (offset 4200H) or II, Versafloppy I, or Tarbell I.

TPM-II - $125 An expanded version of TPM which is fully CP/M 2.2 compatible but still retains the extra features our customers have come to depend on. This version is super FAST. Extended density capability allows over 600K per side on an 8" disk. Available preconfigured for Versafloppy II (8" or 5"), Epson QX-10, Osborne II or TRS-80 Model II.

CONFIGURATOR I
This package provides all the necessary programs for customizing TPM for a floppy controller which we do not support. We suggest ordering this on single density (8SD).
Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), LINKER ($80), DEBUG I ($80), QED ($150), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$815 Value — NOW $250

CONFIGURATOR II
Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QSAL ($200), QED ($150), ZTEL ($80), TOP II ($100), BUSINESS BASIC ($200) and MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1485 Value — NOW $400

MODEL I PROGRAMMER
This package is only for the TRS-80 Model I. Note: These are the ONLY CDL programs available for the Model I. It includes: TPM I ($80), BUSINESS BASIC ($200), MACRO I ($80), DEBUG I ($80), ZDDT ($40), ZTEL ($80), TOP I ($80) and MODEM ($40)
$680 Value — NOW $175

MODEL II PROGRAMMER
This package is only for the TRS-80 Model II. It includes: TPM-II ($125), BUSINESS BASIC ($200), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QED ($150), ZTEL ($80), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM ($40), MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1445 Value — NOW $375

BASIC I - $50, a 12K+ basic interpreter with 7 digit precision.

BASIC II - $100, A 12 digit precision version of Basic I.

BUSINESS BASIC - $200, A full disk extended basic with random or sequential disk file handling and 12 digit precision (even for TRIG functions). Also includes PRIVACY command to protect source code, fixed and variable record lengths, simultaneous access to multiple disk files, global editing, and more!

ACCOUNTING PACKAGE - $300, Written in Business Basic. Includes General Ledger, Accounts Receivable/Payable, and Payroll. Set up for Hazeltine 1500 terminal. Minor modifications needed for other terminals. Provided in unprotected source form.

MACRO I - $80, A Z80/8080 assembler which uses CDL/TDL mnemonics. Handles MACROs and generates relocateable code. Includes 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical ops, local and global symbols, linkable module generation, and more!

MACRO II - $100, An improved version of Macro I with expanded linking capabilities and more listing options. Also internal code has been greatly improved for faster more reliable operation.

MACRO III - $150, An enchanced version of Macro II. Internal buffers have been increased to achieve a significant improvement in speed of assembly. Additional features include line numbers, cross reference, compressed PRN files, form feeds, page parity, additional pseudo-ops, internal setting of time and date, and expanded assembly-time data entry.

DEVELOPER I
Includes: MACRO I ($80), DEBUG I ($80), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$440 Value — NOW $150

DEVELOPER II
Includes: MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), BUSINESS BASIC ($200), QED ($150), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM SOURCE ($40), ZTEL ($80), and DISASSEMBLER ($80).
$1280 Value — NOW $350

DEVELOPER III
Includes: QSAL ($200), QED ($150), BUSINESS BASIC ($200), ZTEL ($80) and TOP II ($100)
$730 Value — NOW $300

COMBO
Includes: DEVELOPER II ($1280), ACCOUNTING PACKAGE ($300), QSAL ($200) and 6502X ($150)
$1930 Value — NOW $500

LINKER - $80, A linking loader for handling the linkable modules created by the above assemblers.

DEBUG I - $80, A tool for debugging Z80 or 8080 code. Disassembles to CDL/TDL mnemonics compatible with above assemblers. Traces code even through ROM. Commands include Calculate, Display, Examine, Fill, Goto, List, Mode, Open File, Put, Set Wait, Trace, and Search.

DEBUG II - $100, A superset of Debug I. Adds Instruction Interpreter, Radix change, Set Trap/Conditional display, Trace options, and Zap FCB.

6502X - $150, A 6502 cross assembler. Runs on the Z80 but assembles 6502 instructions into 6502 object code! Similar features as our Macro assemblers.

QSAL - $200, A SUPER FAST Z80 assembler. Up to 10 times faster than conventional assemblers. Directly generates code into memory in one pass but also to offset for execution in its own memory space. Pascal like structures: repeat...until, if...then...else, while...do, begin...end, case...of. Multiple statements per line, special register handling expressions, long symbol names, auto and modular assembly, and more! This one uses ZILOG Mnemonics.

QED - $150, A screen editor which is both FAST and easy to learn. Commands include block delete, copy, and move to a named file or within text, repeat previous command, change, locate, find at start of line, and numerous cursor and window movement functions. Works with any CRT having clear screen, addressable cursor, clear to end of line, clear to end of screen, and 80X24.

ZTEL - $80, An extensive text editing language and editor modelled after DEC's TECO.

ZEDIT - $50, A mini text editor. Character/line oriented. Works well with hardcopy terminals and is easy to use. Includes macro command capability.

TOP I - $80, A Text Output Processor for formatting manuals, documents, etc. Interprets commands which are entered into the text by an editor. Commands include justify, page number, heading, subheading, centering, and more.

TOP II - $100, A superset of TOP I. Adds: embedded control characters in the file, page at a time printing, selected portion printing, include/merge files, form feed/CRLF option for paging, instant start up, and final page ejection.

ZDDT - $40, This is the disk version of our famous Zapple monitor. It will also load hex and relocatable files.

ZAPPLE SOURCE - $80, This is the source to the SMB ROM version of our famous Zapple monitor. It can be used to create your own custom version or as an example of the features of our assemblers. Must be assembled using one of our assemblers.

MODEM - A communication program for file transfer between systems or using a system as a terminal. Based on the user group version but modified to work with our SMB board or TRS-80 Models I or II. You must specify which version you want.

MODEM SOURCE - $40, For making your own custom version. Requires one of our Macro Assemblers.

DISASSEMBLER - $80, Does bulk disassembly of object files creating source files which can be assembled by one of our assemblers.

HARDWARE

S-100 — SMB II Bare Board $50. "System Monitor Board" for S-100 systems, 2 serial ports, 2 parallel ports, cassette interface, 4K memory (ROM, 2708 EPROM, 2114 RAM), and power on jump. When used with Zapple ROM below, it makes putting a S-100 system together a snap.
Zapple ROM $35. Properly initializes SMB I/II hardware, provides a powerful debug monitor.

IBM PC — Big Blue Z80 board $595. Add Z80 capability to your IBM Personal Computer. Runs CP/M programs but does not require CP/M or TPM. Complete with Z80 CPU, 64K add on memory, serial port, parallel port, time and date clock with battery backup, hard disk interface, and software to attach to PC DOS and transfer programs. Mfr'd by QCS.
50% Discount on all CDL software ordered at the same time as a Big Blue (and for the Big Blue).

APPLE II — Chairman Z80 $345. Add Z80 capability to your Apple II/II Plus computer. Runs CP/M programs with our more powerful TPM. Includes 64K memory add on (unlike the competition this is also useable by the 6502/DOS as well as the Z80), TPM, QSAL assembler, QED Screen Editor, and Business Basic. Mfr'd by AMT Research.
Apple Special $175. Buy the Apple Z80 Developer at the same time as the "Chairman" and pay only $175 instead of $325.

APPLE Z80 DEVELOPER
Includes: 6502X ($150), MACRO II ($100), MACRO III ($150), QSAL ($200), QED ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), ZDDT ($40) and BUSINESS BASIC ($200)
VALUE: $1250 — NOW $325
$175 when purchased with AMT "Chairman" Board

DISK FORMATS

When ordering software specify which disk format you would like.

CODE	DESCRIPTION
8SD	8" IBM 3740 Single Density (128 bytes/26 sectors/77 tracks)
8DD	8" Double Density (256 bytes/26 sectors/77 tracks)
8XD	8" CDL Extended Density (1024 bytes/8 secter/77 traceks - 616K)
5SD	5.25" Single Density (TRS80 Model I, Versafloppy I, Tarbell I)
5EP	5.25" Epson Double Density
5PC	5.25" IBM PC Double Density
5XE	5.25" Xerox 820 Single Density
5OS	5.25" Osborne Single Density
5ZA	5.25" Z80 Apple (Softcard compatible)

TPM INFO When ordering TPM I or II, in addition to Disk Format, please specify one of the following codes:

	CODE	DESCRIPTION
TPM I:	NSSD/H	North Star Single Density for Horizon I/O
	NSSD/Z	North Star Single Density for Zapple I/O
	NSDD/H	North Star Double Density for Horizon I/O
	NSDD/Z	North Star Double Density for Zapple I/O
	TRS80-I	TRS-80 Model I (4200H Offset)
	TRS80II	TRS-80 Model II
	VI8	Versafloppy I 8"
	VI5	Versafloppy I 5.25"
TPM-II:	VII8	Versafloppy II 8" (XD)
	VII5	Versafloppy II 5.25"
	TRS80II	TRS-80 Model II (XD)

Prices and Specifications subject to change without notice.
TPM, Z80, CP/M, TRS80 are trademarks of CDL, Zilog, DRI and Tandy respectively.

ORDERING INFORMATION:

VISA/MasterCard/C.O.D.
Call or Write With Ordering Information....

OEMS:

Many CDL products are available for licensing to OEM's. Write to Carl Galletti with your requirements.

Dealer Inquiries Invited.

For Phone Orders ONLY Call Toll Free...
1-(800) 458-3491 (Except Pa.)
Ask For Extension #15

For information and Tech Queries call
(609) 599-2146

Computer Design Labs

342 Columbus Avenue/Trenton, NJ 08629

Circles (Listing continued, text begins on page 18)

Listing Two

```
08D5 8D 2E 09              STA HL
08D8 AD 38 09              LDA XYH
08DB E9 00                 SBC #0
08DD 8D 2D 09              STA HH
08E0             ;
08E0 AD 39 09              LDA XYL          ;V<-[XY-XCRD]
08E3 38                    SEC
08E4 ED 35 09              SBC XCRDL
08E7 8D 31 09              STA V
08EA             ;
08EA 20 2C 08              JSR PLOTFOUR     ;PLOT REMAINING POINTS
08ED             ;
08ED AD 3C 09              LDA YMID         ;DIVISOR<-[YMID-YCRD]
08F0 38                    SEC
08F1 ED 3D 09              SBC YCRD
08F4 85 1A                 STA DIVISOR
08F6             ;
08F6 AD 37 09              LDA XMIDL        ;DIVIDEND<-[XMID-XCRD]*256
08F9 38                    SEC
08FA ED 35 09              SBC XCRDL
08FD             ;
08FD A2 08                 LDX #8           ;BITCT<-8
08FF             ;
08FF A0 00                 LDY #0           ;CLEAR QUOTIENT
0901 84 1B                 STY QUOTIENT
0903             ;
0903 06 1B       DIVIDE1   ASL QUOTIENT
0905 2A                    ROL
0906 C5 1A                 CMP DIVISOR
0908 90 04                 BCC DIVIDE2
090A E5 1A                 SBC DIVISOR
090C E6 1B                 INC QUOTIENT
090E CA          DIVIDE2   DEX
090F D0 F2                 BNE DIVIDE1
0911             ;
0911 AD 3E 09              LDA YCRDF        ;YCRDF<-[YCRDF-QUOTIENT]
0914 38                    SEC
0915 E5 1B                 SBC QUOTIENT
0917 8D 3E 09              STA YCRDF
091A             ;
091A B0 03                 BCS CKX          ;YCRDY< -1?
091C CE 3D 09              DEC YCRD         ;YES, YCRD<-[YCRD-1]
091F             ;
091F EE 35 09    CKX       INC XCRDL        ;XCRD<<-[XCRD+1]
0922 D0 03                 BNE CKX1
0924 EE 34 09              INC XCRDH
0927             ;
0927 CE 30 09    CKX1      DEC R2           ;TALLY R2
092A 10 8D                 BPL NXPOINT      ;REPEAT UNTIL XCRD=XMID
092C 60                    RTS
092D             ;
092E             HH        DFS 1            ;HORIZONTAL PLOT VALUE
092F             HL        DFS 1
```

(Continued on page 28)

USE MAGIC/L™ TO WRITE JUST ONE PROGRAM

Bet you can't stop with one.

For just $295 MAGIC/L™ is yours. And suddenly you find you've bought more than a language. MAGIC/L is an entire interactive environment; an assembler, a compiler, an editor, an I/O package, and a system call facility are all wrapped up and delivered. And now MAGIC/L is available for CP/M-based computers.

MAGIC/L is easy to learn. It has syntax similar to C and Pascal, and because it's extensible as well as interactive, it dramatically increases productivity.

Program development features include a built-in text editor, command line recall, CCP, STAT, and PIP command emulation, and the ability to store keyboard dialog on disks.

Key language features include: CHAR, INTEGER, LONG, REAL, and String data types; record structures similar to the STRUCT facility in C; and a complete I/O package that can provide random access, variable length I/O to any CP/M file.

And MAGIC/L offers great portability. Source code which runs on CP/M can be compiled unmodified and run on any other processor.

Typical applications include hardware interfacing, process control, games creation, interactive graphics and image processing. MAGIC/L has made programming easier for DEC, 68000, and Data General users. Now it's working for CP/M users too.

MAGIC/L provides everything you need to write a complete program. But the only way to be convinced is to try it yourself. Send us your $295 check or money order—we also accept MasterCard and VISA—we'll send MAGIC/L for CP/M to you at once. A full money back guarantee is part of the package. Once you've sampled that first program, you'll have to try another . . . and another . . . and another. MAGIC/L . . . it's more than a language.

MAGIC/L . . . It's more than a language

LOKI ENGINEERING, INC.
55 Wheeler St., Cambridge, MA 02138 (617) 576-0666

Circle **no. 60** on reader service card.

Circles (Listing continued, text begins on page 18)

(Listing continued, text begins on page 18)

Listing Two

```
0930     R         DFS 1          ;RADIUS
0931     R2        DFS 1          ;HOLDS R/SQR(2)
0932     V         DFS 1          ;VERTICAL PLOT
0933     X2H       DFS 1          ;HOLDS XMID*2
0934     X2L       DFS 1
0935     XCRDH     DFS 1          ;X COORDINATE
0936     XCRDL     DFS 1
0937     XMIDH     DFS 1          ;HORIZONTAL CENTER
0938     XMIDL     DFS 1
0939     XYH       DFS 1          ;HOLDS XMID+YMID
093A     XYL       DFS 1
093B     Y2H       DFS 1          ;HOLDS YMID*2
093C     Y2L       DFS 1
093D     YMID      DFS 1          ;VERTICAL CENTER
093E     YCRD      DFS 1          ;Y COORDINATE
093F     YCRDF     DFS 1          ;FRACTIONAL PART OF YCRD
093F     ;
001A     DIVISOR   EPZ $1A
C052     FULL      EQU $C052
C050     GRAPHICS  EQU $C050
00E4     HCOLOR    EPZ $E4
C057     HIRES     EQU $C057
F457     HPLOT     EQU $F457      ;APPLESOFT HIRES PLOT
00E6     HPAGE     EPZ $E6
00E1     LASTHH    EPZ $E1        ;HORIZ COORD OF LAST HPLOT
00E0     LASTHL    EPZ $E0
00E2     LASTV     EPZ $E2        ;VERT COORD OF LAST HPLOT
C054     PAGE1     EQU $C054
001B     QUOTIENT  EPZ $1B
093F     ;
093F               END

***** END OF ASSEMBLY
```

End Listing Two

(Listing Three begins on page 30)

Programmers...Now you can "C" into your programs!

C Compiler

Runs on the IBM® Personal Computer or MSDOS® compatible computers. Provides features and flexibility needed for both systems and application programming. Makes it possible to test programs at the C source level. The software program features full documentation, generates fast code, and has a standard C function library.

- All C language elements for operations on characters, signed and unsigned integers, structures, and unions
- Outputs IBM PC assembler source code
- PL/M compatibility mode
- Embedded shell in compiled output supports argc, argv, and I/O redirection
- Complete preprocessor
- Allows more than 64K of code; extended pointers allow access to more than 64K of data

c-window™ Source Level Debugger

Designed for sophisticated programmers who want more capability than inserting "printf" statements or using assembler level debugging . . . new programmers who want to better understand how C operates . . . trainers who teach the C language. It helps you find the precise statement in a program where an error occurs and observe when data values are changed in a program.

- True source level single step
- Auto and external variable display and modification in a variety of formats
- Full C expression evaluation and recursion
- Varied breakpoints
  - —at function name and line number
  - —sticky and non-sticky
  - —at function whenever an expression evaluates as TRUE
- Multiple variable display as program is tested

c-systems

The following is an excerpt from a typical c-window debugging session. The ">>" is the c-window prompt, and the underscored text is the operator input. The text on the right is a brief description of the operations taking place.

```
entry at main line 15
>>bs addline, 58

>>g
break at addline line 58
>>d ptr
bae
>>d ptr = root
f02
>>d ++ptr
f14
>>ds ptr->symname
xyzzy
>>cs 1 d i

>>s
step at addline line 59

<1> d i
11
>>d i==j
1
>>bx addline,2,i<j&& k==0
```

- c-window entry prompt showing function name and line number
- A breakpoint is set in function addline, line 58.
- The "go" command starts execution.
- The breakpoint is reached.
- The value of variable "ptr" is displayed.
- The variable "ptr" is set to the value of the variable "root".
- The variable "ptr" is incremented to the next entry in the table.
- Display the string at the member "symname" in the structure pointed to by "ptr".
- Command set. The variable "i" will be displayed for each break in execution.
- Single step.
- c-window displays the next line to be executed.
- c-window executes the automatic command, showing the value of "i" to be 11 hex.
- Test if "j" has the same value as "i".
- They are equal.
- Set an expression break. Once execution begins, c-window will break execution on the second occurence of the expression "i<j&&k==0" evaluating non-zero in the function "addline".

For the finest time-saving software contact:

c-systems We make programming easier

P.O. Box 3253 • Fullerton, CA 92634 • (714) 637-5362

Name ______________________

Address ______________________

____

I would like to order:

- ☐ C Compiler. $195.00
- ☐ c-window (requires C Compiler) $195.00
- ☐ Demo package (includes diskette, manual) price applicable toward purchase of above . . $45.00

Require 128K and two disk drives.
c-window is a trademark of C-Systems. IBM is a trademark of IBM Corp. and MSDOS is a trademark of MicroSoft.
Prices subject to change without notice.

Circle no. 26 on reader service card.

Circles (Listing continued, text begins on page 18)

Listing Three

```
10    '*** CIRCLE PLOT ***
20    '
30    INPUT "CENTER, RADIUS"; CX,CY,R: X=R: Y=0: A=-2*X+1:
      B=1: GOSUB 70: GOTO 30
40    '
50    'PLOT A POINT IN EACH OCTANT
60    '
70    PSET(X+CX,Y+CY): PSET(Y+CX,X+CY): PSET(-Y+CX,X+CY):
      PSET(-X+CX,Y+CY): PSET(-X+CX,-Y+CY): PSET(-Y+CX,-X+CY):
      PSET(Y+CX,-X+CY): PSET(X+CX,-Y+CY)
80    '
90    'COMPUTE NEXT POINT.  F IS X^2+Y^2-R^2, A IS THE CHANGE
100   'IN X^2 WHEN X IS DECREMENTED BY 1, AND B IS THE CHANGE
110   'IN Y^2 WHEN Y IS INCREMENTED BY 1.  F IS NOT ALLOWED TO
120   'EXCEED  R; EQUIVALENTLY, THE POINT (X,Y) IS KEPT WITHIN
130   'A DISTANCE R+1/2 OF THE CIRCLE CENTER.  THE ALGORITHM
140   'IS DONE WHEN THE CHANGE IN Y^2 REACHES THE NEGATIVE OF
150   'THE CHANGE IN X^2 ( B>=-A ).
160   '
170   IF B>=-A THEN RETURN ELSE Y=Y+1: F=F+B: IF F>R THEN
      F=F+A: A=A+2: X=X+1
180   B=B+2: GOTO 70
```

End Listing Three

Circles

Listing Four

```
10 DEFINT A-Z
20 PRINT CHR$(26) 'CLEAR DUMB TTY SCREEN
50 FOR I=1 TO 11
55 AE=I*2 'WIDTH OF ELLIPSE
56 BE=I*1 'HEIGHT OF ELLIPSE
57 XC=I*4+1 'CENTER.X OF ELLIPSE
58 YC=I*1 'CENTER.Y OF ELLIPSE
60 GOSUB 1060 'PLOT A CIRCLE
70 NEXT I 'PLOT 11 CIRCLES
998 END
1050 '****** CIRCLE SUBROUTINE
1060 XF=0                   'INIT X-OFFSET
1070 YF=BE                  'INIT Y-OFFSET
1080 XD=BE*BE               'INIT COMPUTATION OF X-SQUARED
1090 YD=(2*BE-1)*AE*AE      'INIT COMPUTATION OF Y-SQUARED
1100 DX=2*BE*BE             'DEFINE DELTA-(X-SQUARED)
1110 DY=2*AE*AE             'DEFINE DELTA-(Y-SQUARED)
1120 ER=0                   'INIT ERROR (I.E. ER=AE^2*BE^2-XF^2*BE^2-YF^2*AE^2)
1130 GOSUB 1260             'PLOT THE FOUR POINTS
1140 TX=ER+XD
```

```
        : TY=ER-YD
        : TB=ER+XD-YD
1150 IF ABS(TX)>=ABS(TY) OR ABS(TX)>=ABS(TB) THEN 1170
1160 XF=XF+1
        : ER=TX
        : XD=XD+DX
        : GOTO 1220
1170 IF ABS(TY)>=ABS(TX) OR ABS(TY)>=ABS(TB) THEN 1190
1180 YF=YF-1
        : ER=TY
        : YD=YD-DY
        : GOTO 1220
1190 IF ABS(TB)>=ABS(TX) OR ABS(TB)>=ABS(TY) THEN 1210
1200 XF=XF+1
        : YF=YF-1
        : ER=TB
        : YD=YD-DY
        : XD=XD+DX
        : GOTO 1220
1210 PRINT"OOPS"; 'IF HERE THEN THERE IS A BUG.
1220 GOSUB 1260 'PLOT THE POINTS
1230 IF  YF<>0 THEN 1140
1240 RETURN
1250 '****** ROUTINE TO PLOT FOUR POINTS AT ONCE
1260 XP=XC+XF
        : YP=YC+YF
        : GOSUB 1320
1270 XP=XC+XF
        : YP=YC-YF
        : GOSUB 1320
1280 XP=XC-XF
        : YP=YC+YF
        : GOSUB 1320
1290 XP=XC-XF
        : YP=YC-YF
        : GOSUB 1320
1300 RETURN
1310 '****** ROUTINE TO PLOT A POINT ON A DUMB TERMINAL
1320 C1=YP+32
        : C2=XP+32
1330 IF YP<0 OR YP>23 OR XP<0 OR XP>79 THEN 1360
1340 PRINT CHR$(27);CHR$(61);CHR$(C1);CHR$(C2);"*";
1350 RETURN
1360 PRINT "POINT OUT OF BOUNDS"
        : STOP
```

End Listing Four

Cursor Control for Dumb Terminals

A friend of mine asked me one day, "Why should I purchase a new CRT terminal when I can buy a surplus unit for less than half the price?"

"Because," I answered, "that so-called bargain of yours is what we used to call a 'dumb' terminal. Unlike today's marvels, it has only the most basic of cursor control functions: home, line feed, carriage return, backspace, and skip (the skip function moves the cursor across the screen one column at a time without deleting any currently at that position). It is incapable of direct cursor addressing, which is required by most of the software available on the market today. In short, it is not a bargain; it is an antique."

"What," my friend asked, "should I do with this antique I just bought?"

The solution to his problem turned out to be surprisingly simple and should be of interest to anyone owning a terminal without direct cursor addressing and a microcomputer system running CP/M. (It should also interest those wanting to connect more esoteric peripherals than terminals or printers to their systems, if only to illustrate the level of complexity that can be supported.) Only the software need be modified. Even then the CP/M operating system remains absolutely standard and is still capable (assuming the IOBYTE feature has been implemented) of supporting a second terminal with direct cursor addressing without further modifications. What more could you ask for?

A terminal with direct cursor addressing traps a predefined string of ASCII characters sent from the computer to the terminal and uses the information contained in this string to position the cursor on the screen. (The string itself, of course, is not displayed.) As an example, the Televideo 950 terminal responds to the ASCII string "ESC=yx," where y is an ASCII character from "SP" (32 decimal) to "7" (55 decimal) that positions the cursor vertically and x is an ASCII character from "SP" to "o" (111 decimal) that positions the cursor horizontally. To emulate this feature, a "dumb" terminal should be able to respond to a similar string of ASCII characters and position its cursor accordingly.

While it would be nice to have the terminal perform this function through its own hardware, this is obviously impractical for a large number of users; redesigning and rewiring old printed circuit boards without schematics is a task few care to tackle! What remains is to modify the software, specifically CP/M's BIOS (Basic Input/Output System), which provides the interface between the terminal and the host computer. The CP/M operating system can then monitor what the running program is sending through the BIOS to the terminal.

When an ASCII "ESC" character is sent to the modified BIOS (from, say, a full-screen word processing program), it will store this character temporarily rather than output it through the serial port to the terminal. When the next character sent by the program is received, the BIOS will check to see if it is an ASCII "=" character. If it isn't, the BIOS will output both the stored "ESC" character and the received character to the terminal, then continue to output characters received from the program until the next "ESC" character arrives; otherwise, it will recognize the two characters as the start of a direct cursor positioning command, discard them, and wait for the next character.

Once this character is received, the BIOS will decode it, output a single ASCII control character to home the cursor on the terminal, and output the required number of line feeds to position the cursor vertically. Then it will trap the next character sent by the program, decode it, and output the required number of skip control characters to position the cursor horizontally. Once the cursor is positioned where required, everything returns to normal: all ASCII characters sent to the BIOS by the program will be sent to the terminal after being monitored for an "ESC" character and the possible start of another direct cursor positioning command.

It sounds complicated, but it is simple to program and works beautifully as long as you operate the terminal at 9600 baud. At this speed the cursor will take, in the worst case, approximately one-tenth of a second to traverse the screen (from the home position to the 24th row and 80th column). The motion of the cursor is perceptible but not objectionable; a slower baud rate, however, is an invitation to crossed eyeballs from watching the cursor bounce around the screen!

The entire software routine can be written as a peripheral driver and added to your BIOS in exactly the same way as you would add a printer or modem to your system.

If your version of CP/M doesn't include the IOBYTE feature, then the accompanying code (Listing One, page 33) should be substituted for your existing terminal driver (usually the physical device TTY: or CRT: in the BIOS source code). Once this is done, your version of CP/M will be customized for the "dumb" terminal being used as its console device. When a newer model terminal is purchased, the BIOS must be customized once again to support it as the console device instead of the old terminal.

Fortunately, most commercial versions of CP/M do include the IOBYTE feature. In this case, the accompanying code should be placed in whatever physical I/O device driver routine isn't being used (usually the user-defined console device UC1:). By leaving the TTY: or CRT: physical device code (whichever currently supports the computer's terminal device) intact or by modifying it to support the terminal you eventually intend to purchase, you can switch between a "dumb" terminal and a modern direct-cursor-addressable terminal simply by using the STAT CON:=UC1: and STAT CON:=TTY: (or STAT CON:=CRT:, as applicable) commands. You can also modify your BIOS to initially set the IOBYTE so that CON:=UC1: after each cold boot.

Because the procedure for modifying CP/M's BIOS may vary, depending on the idiosyncracies of your particular system, I will not describe it in this article. If you haven't interfaced a peripheral to your system before, I suggest that you have your vendor or an experienced friend perform the modifications indicated in Listing One; otherwise, you should consult some of the excellent books on CP/M that are available (the finest I know of is David Cortesi's *Inside CP/M – A Guide for Users and Programmers*, CBS College Publishing, 1982).

By the way, some excellent bargains are available on the surplus computer equipment market – if you know exactly what you are buying. Otherwise, caveat emptor, or "let the buyer beware!" DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 197.**

by Ian Ashdown

Ian Ashdown, byHeart Software, 2-2016 West First Avenue, Vancouver, B.C., V6J 1G8, Canada.

Cursor Control Listing (Text begins on page 32)

```
; ****************************************************************
; *                                                              *
; *                  CP/M BIOS Modifications                     *
; *                                                              *
; *     Author: Ian Ashdown                                      *
; *             byHeart Software                                 *
; *             2 - 2016 West First Avenue                       *
; *             Vancouver, B.C. V6J 1G8                          *
; *                                                              *
; *     Date:   February 10th, 1983                              *
; *                                                              *
; ****************************************************************
;
; The following code has been added to allow the ANAND II "dumb"
; terminal to emulate a direct-addressable cursor terminal
; (specifically the TELEVIDEO 950). It does this by trapping an
; "ESC=yx" control code from a program and interpreting "y" as
; the vertical row address (where 20H is row #1) and "x" as the
; horizontal column address of the cursor (where 20H is column
; #1). The BIOS then homes the cursor and outputs as many line
; feeds and skips as is required to position the cursor without
; changing the current screen display. At 9600 baud the worst
; case transit time of the cursor (from "home" to the 24th row
; and 80th column) is approximately 1/10 second.
;
; ************************* EQUATES ****************************
;
; NOTE: The following equates are independent of the system used.
;       However, they may already be defined in your BIOS source
;       code.
;
LF      EQU     0AH     ;ASCII line feed
CR      EQU     0DH     ;ASCII carriage return
ESC     EQU     1BH     ;ASCII escape
;
; NOTE: The following equates are specific to my California
;       Computer System's serial I/O port and the control codes
;       recognized by the ANAND II terminal. Replace them
;       with those required by your system.
;
RESET   EQU     19H     ;ANAND II "home cursor" control char.
SKIP    EQU     14H     ;ANAND II "cursor forward" control char.
SDATA   EQU     20H     ;SERIAL DATA PORT
SLSTAT  EQU     SDATA+5 ;Serial line status port
RXRDY   EQU     00000001B  ;Receive data available bit
TXMTY   EQU     00100000B  ;Transmit buffer empty bit
;
; ********** NEW PHYSICAL DEVICE UC1: DRIVER ROUTINE ***********
;
; I/O DRIVERS FOR THE 8250 ASYNC COMM ELEMENT
; (PHYSICAL DEVICE UC1:)
;
```

(Continued on next page)

Cursor Control Listing (Listing continued, text begins on page 32)

```
; NOTE: The following CUST1: and CUSI1: subroutines are vendor-
;       dependent code (see COMMENTARY at the end of this
;       listing), and should be replaced with the TTY: or CRT:
;       driver code specific to your system.
;
CUST1:  IN      SLSTAT  ;Get 8250 line status
        ANI     RXRDY   ;See if receive data available
        RZ              ;Return if not
        ADI     0FFH AND NOT RXRDY      ;Flag that data is
        RET                             ;available
;
CUSI1:  CALL    CUST1   ;Get 8250 line status
        JZ      CUSI1   ;Loop until data is in
        IN      SDATA   ;Read the data
        RET
;
; NOTE: The following subroutines CUSO1: through HORZOUT:
;       inclusive are independent of the system used.
;
CUSO1:  LDA     ESCPTR  ;Get escape sequence pointer
        CPI     0       ;Jump to TESTCH if this isn't an
        JZ      TESTCH  ;escape sequence
        CPI     1       ;Jump to TESTEQU if this is an escape
        JZ      TESTEQU ;sequence
        CPI     2       ;Jump to VERTIN if true (character is
        JZ      VERTIN  ;row address of cursor)
        JMP     HORZIN  ;Jump to HORZIN (character is column
                        ;address of cursor)
TESTCH: MOV     A,C     ;Get character from register C
        CPI     ESC     ;Jump to NOTESC if it isn't an ASCII ESC
        JNZ     NOTESC
        MVI     A,1     ;Set escape sequence pointer to 1 to
        STA     ESCPTR  ;indicate ASCII ESC character
        RET
NOTESC: CALL    CUSOUT  ;Output character
        RET
;
TESTEQU:MOV     A,C     ;Get character from register C
        CPI     '='     ;Jump to NOTEQU if it isn't an ASCII "="
        JNZ     NOTEQU
        MVI     A,2     ;Set escape sequence pointer to 2 to
        STA     ESCPTR  ;indicate ASCII "=" character
        RET
NOTEQU: MVI     A,0     ;Set escape sequence pointer to 0 (false
        STA     ESCPTR  ;alarm)
        MVI     A,ESC   ;Output previous ASCII ESC character
        CALL    CUSOUT
        MOV     A,C     ;Get current character from register C
        CALL    CUSOUT  ;Output it
        RET
;
```

(Continued on page 36)

Cursor Control Listing (Listing continued, text begins on page 32)

```
VERTIN: MOV     A,C      ;Get character from register C
        SUI     1FH      ;Subtract cursor address bias (20H is row
                         ;number 1) to get row address counter
        MOV     C,A      ;Store row address counter in register C
        MVI     A,3      ;Set escape sequence pointer to 3
        STA     ESCPTR
        MVI     A,RESET  ;Output "reset cursor" control character
        CALL    CUSOUT
VERTOUT:DCR     C        ;Decrement row address
        RZ               ;Return if done
        MVI     A,LF     ;Output a line feed
        CALL    CUSOUT
        JMP     VERTOUT  ;Loop until done
;
HORZIN: MOV     A,C      ;Get character from register C
        SUI     1FH      ;Subtract cursor address bias (20H is
        MOV     C,A      ;column number 1) to get column counter
        XRA     A        ;Set escape sequence pointer to 0
        STA     ESCPTR
HORZOUT:DCR     C        ;Decrement column counter
        RZ               ;Return if done
        MVI     A,SKIP   ;Output a "cursor skip" control character
```

```
         CALL     CUSOUT
         JMP      HORZOUT ;Loop until done
;
ESCPTR   DB       0       ;Escape sequence pointer
;
; NOTE:  The following CUSTOUT: subroutine is vendor-dependent
;        code (see COMMENTARY at the end of this listing), and
;        should be replaced with the TTY: or CRT: driver code
;        specific to your system.
;
CUSOUT:  PUSH     PSW     ;Save character
CUSOUT1:CALL      CUSOST  ;Get 8250 line status
         JZ       CUSOUT1 ;Wait until one of the registers empties
         POP      PSW     ;Recall character
         OUT      SDATA   ;Output the data
         RET
;
CUSOST:  IN       SLSTAT  ;Get 8250 line status
         ANI      TXMTY   ;Isolate TX buffer empty bit
         RZ               ;Return if not empty
         ADI      OFFH AND NOT TXMTY       ;Flag the empty state
         RET
;
; *********************** DELETED CODE ***************************
;
```

(Continued on next page)

Cursor Control Listing (Listing continued, text begins on page 32)

(Listing continued, text begins on page 32)

```
; The following code has been deleted:
;
;CUSI1: EQU      IOER     ;UNASSIGNED USER CONSOLE (INPUT)
;CUSO1: EQU      IOER     ;UNASSIGNED USER CONSOLE (OUTPUT)
;CUST1: EQU      IOER
;
; ************************ COMMENTARY ***************************
;
; The following California Computer Systems vendor code for a
; typical serial interface to a terminal has been added for
; information only as an example to show where the I/O portion
; of the above UC1: code was derived from. Most CP/M TTY: and
; CRT: physical device drivers will be similar to this code.
;
; I/O DRIVERS FOR THE 8250 ASYNC COMM ELEMENT
; (PHYSICAL DEVICE TTY:)
;
;TTST:   IN       SLSTAT   ;GET 8250 LINE STATUS
;        ANI      RXRDY    ;SEE IF RECEIVE DATA AVAILABLE
;        RZ                ;RETURN IF NOT
;        ADI      OFFH AND NOT RXRDY      ;FLAG THAT DATA IS
;        RET                              ;AVAILABLE
;
;TTYIN:  CALL     TTST     ;GET 8250 LINE STATUS
;        JZ       TTYIN    ;LOOP UNTIL DATA IS IN
;        IN       SDATA    ;READ THE DATA
;        RET
;
;TTOST:  IN       SLSTAT   ;GET 8250 LINE STATUS
;        ANI      TXMTY    ;ISOLATE TX BUFFER EMPTY BIT
;        RZ                ;RETURN IF NOT EMPTY
;        ADI      OFFH AND NOT TXMTY      ;FLAG THE EMPTY STATE
;        RET
;
;TTYOUT:CALL      TTOST    ;GET 8250 LINE STATUS
;        JZ       TTYOUT   ;WAIT UNTIL ONE OF THE REGISTERS EMPTIES
;        MOV      A,C      ;MOVE THE DATA OVER
;        OUT      SDATA    ;OUTPUT THE DATA
;        RET
;
; *********************** END OF LISTING ************************
```

End Listing

GREAT DRIVES. GREAT PRICES.

Floppy Disk Services offers both.

We supply the industry with high quality disk drives and peripherals. And our prices are some of the best you'll see. Floppy Disk Services is a contracted dealer for Siemens, Tandon and Shugart. Starting with their drives, we design complete packages to give you the system capacity you're looking for and the dependability you need.

We offer add-on drives for IBM, Radio Shack, Heath, Apple and most other microcomputers, all at a significant savings to you. Check the examples below and you'll see what we mean.

Apple II Add on drives	$300.00
Apple 8 inch controller	315.00
Apple dual 8 inch system w/controller	1165.00
FDD-100-5b 'flippy' exact HEATH add on	215.00
FDD-200-5 double sided 40 track drive	250.00
SA 455 half hgt DS/DD 48TPI ... $295.00	2 for $245.00 each
SA 465 half hgt DS/DD 96TPI ... $350.00	2 for $295.00 each
FDD-221-5 5ms step 80 track DD/DS	330.00
FDD-100-8d 8" single side DD drive .... $240.	2 / $225. each
FDD-200-8p Double sided 8" drive .... $340.	2 / $325. each
TM-100-2 (IBM)	265.00
SA-860 DS/DD Half Hgt 8" $495.00	2 for $445.00 each
System packages available for all drives	
Dual 8 inch system with EVERYTHING	750.00*
Dual double sided 8 inch system	950.00*
Single 5¼ Heath or MOD I Add on w/case	265.00*
Dual 5¼ Heath or MOD I	505.00*
10mb Hard Disk for any computer	2300.00*
Magnolia controller, allows any combo 8 and 5¼ inch drives to be added to your H88 or H89	525.00

* 8 inch systems are fully assembled and tested, however the drives are shipped separately from the case to comply with UPS weight restrictions. All 5¼ inch systems come assembled and tested. Prices and specs subject to change without notice.

There's much more. If you don't see what you want, give us a call between 9 am and 5 pm (ET). Chances are we'll have what you need at your price.

Featuring the Shugart Thinline Series.

Floppy Disk Services carries the new Shugart Thinline Dual 8 Inch Drive (Model SA-860). It's a double-density drive, with dual head for a storage capacity of 1.25 megabytes per drive. The system includes our custom cabinets, comes fully assembled and tested, and uses the Power One CP-206 supply, the standard of the industry. Special options available: external drive select using DIP switches, and cable connector on the rear of the cabinet for ease in connecting to your system. Floppy Disk Services designs and builds these systems from the ground up to maximize efficiency and minimize space requirements. Available with Tandon Drives also.

Custom Enclosures. At Floppy Disk Services, we also sell disk drive enclosures, designed by our own experts to be functional and attractive. And our quantity pricing is so competitive, we invite dealers and group purchasers to call.

Thinline/Half Height Specials. Floppy Disk Services has just contracted with Shugart, and we're carrying their half-height 5¼ & 8" systems. If you would like some unbelievable prices on SA-455, SA-465, SA-860's and others, call today! We are legitimate contracted dealers—no middle-man.

Due to production deadlines, prices in this ad are 2 months old, so we encourage you to call us for current prices and new product info.

PAYMENT POLICY — We accept MasterCard, VISA, personal checks & MO. We reserve the right to wait 10 working days for personal checks to clear your bank before we ship. All shipping standard UPS rates plus shipping & handling. NJ residents must add 6% tax.

Free Catalogue. If you'd like to receive our Catalogue of Disk Drives and Peripherals, just call or write — we'll mail your copy immediately. And if you want to talk with an expert about getting more out of your system, we'll be happy to help.

Toll Free Order Line **(800) 223-0306**
Tech Help or Info. **(609) 799-4440**

FLOPPY DISK SERVICES™ INC.

741 Alexander Road Princeton, NJ 08540

Circle **no. 39** on reader service card.

Dysan's Digital Diagnostic Diskette

Dilemma: You've been seeing more flexible disk errors lately, especially when swapping disks between your drives. But they seem to work properly most of the time, so you struggle on, tolerating the occasional loss of your data, frustration, and wasted time. After all, you're not sure anything is wrong with the drives; to find out, you'd have to remove one from the system and ship it off to a stranger who probably makes money five times as fast as you do. If the problem persisted after the drive came back, would you send another one off to be checked?

Deliverance: Now, for less than what you'd probably spend having one drive professionally checked, you can equip yourself to verify the operation of all your flexible disk drives, anytime you suspect a problem, without disassembling or altering your system! But you'll have to begin now, before you have serious operating problems – partly because you'll have to customize a public domain program to fit the quirks of your terminal and disk controller, but mainly because you'll want to have a "before" picture of your drives' operation to help you recognize any real problems that may appear later.

You thought a drive exerciser and dual-channel triggered sweep oscilloscope were essential for flexible drive alignment? Well, the Dysan Corporation, which makes the Analog Alignment Diskette™ for scope alignment, now offers a Digital Diagnostic Diskette™. Using your own computer system and the program described in this article, you can verify your drives' alignment to within one-half milli-inch of the expensive analog method. Plus, you can check the spindle bearings and clamping mechanism, verify spindle speed, check positioner hysteresis (worn lead screw, bad stepper motor, etc.), verify read/write head azimuth, and check index-hold timing, all under your own operating conditions.

by Loren Amelang

Loren Amelang, Box 24, Philo, CA 95466-0024.

What the DDD Is

The Digital Diagnostic Diskette (DDD™) looks like any other floppy disk. It has ordinary formatting, and conventional data are recorded on it. Figure 1 (page 42) shows a small sample of disk data patterns to scale. You could probably run your disk copy program and duplicate it. But your copy would not be another DDD! The digital alignment concept depends upon having the data fields of the DDD carefully displaced from their proper location or azimuth by known amounts.

For example, let's look at the centering test. Three "alternate offset" tracks on the DDD are offset 7, 8, and 9 milli-inches from the official center of the track (see Figure 2, page 42). (All values given are for 48 track-per-inch drives; if you have 96 or 100 tpi drives, tolerances are tighter and thus offsets are smaller.) All odd-numbered sectors are displaced toward the center of the disk, and even-numbered sectors toward the outside edge. Your head gap is designed to pick up a 12 milli-inch swath of data, so if it is perfectly aligned but the data are offset 7 milli-inches, it "sees" only 5 milli-inches of the data. In analog terms, the amplitude of the signal is reduced drastically; in digital terms, either your drive's electronics can make sense of the weak signal, or you get a read error. As you move to the 9 milli-inch offset track, the width of the recorded data still under a properly positioned head is reduced to 3 milli-inches; if your spindle bearings have .003 inches of play, the signal may disappear entirely at certain points. If your head is dirty, and the accumulated oxide holds it away from proper contact with the media, the already reduced signal amplitude will be cut still further. On the other hand, if your system can reliably read data that are offset 9 milli-inches in either direction, you can be pretty sure that your drive is mechanically sound, its head is reasonably aligned, and its electronics are not being swamped by stray electrical or magnetic fields.

If you want to pin down more exactly the radial alignment of your read/write head, the DDD has "progressive offset" tracks (Figure 3, page 44). Here sector one is offset 1 milli-inch toward the spindle, sector two 1 milli-inch away, sector three 2 milli-inches toward the spindle, and so on out to 13 milli-inches in either direction. (This applies to Revision B; Revision A DDD had ±10 milli-inch maximum offsets.) You read through the odd-numbered and then the even-numbered sectors, noting the first sectors that cannot be read. This information can be translated into a graphic display of your effective head width and its position relative to proper alignment. Electrical or magnetic noise in your environment, "dirt" on your head, or improper head loading will narrow the effective head width. A major advantage of the DDD is that you can run this test on six different tracks spaced across the diskette; analog alignment uses only one central track, with the hope the rest will follow.

The "azimuth rotation" track is another progressive track used to check whether your read/write head gap is adversely skewed relative to the data track. Here data are recorded on the radial centerline of the track, but the head azimuth is progressively rotated away from normal (Figure 4, page 46). If read errors occur symmetrically, your effective head azimuth is correct. If your system fails in one direction before the other, you may improve performance by altering your head azimuth toward the earlier angle. Again, a decrease in azimuth range could indicate a dirty head or environmental noise.

Other recorded DDD tracks contain a special formatting pattern for timing the passage of the index hole past your sensor and for detecting how long it takes your head to settle into position after it is loaded. Finally, some tracks are marked "user area," where you can store the program you'll need to read the DDD and display the results. *Don't* try to do this under CP/M! You'll overwrite other parts of the DDD and destroy it! If you are going to make use of the "user area," you'll need a "disk doctor" program that can write to specific tracks of the disk, and you'll need some way to load and run what you've stored without involving CP/M.

How does Dysan Corporation manage to create a disk with precisely misaligned information on it? To begin, they take their premium diskettes and screen them: any alignment diskette must meet high standards for dimensional stability and uniformity of the recording surface. The DDD must be error-free, not only in the normal tracks but "in the cracks" as well. Critical information is going to be recorded completely off track for some of the tests.

Several million dollars went into the computer-controlled, laser-interferometer-calibrated trackwriters and the environ-

mentally standardized room that houses them. Once the blank diskettes have acclimated to the standard environment, they are individually written; the track-writers actually move the write heads to all those different off-track positions. These same machines write the Analog Alignment Diskettes used by manufacturers and calibration services. The same precise standards are met, both for the media and for the recorded information. The only advantage of analog alignment is that it is not limited to resolving the 1 milli-inch steps of the DDD but can get closer to the exact center of track. Of course, the DDD has the advantage of being able to check alignment on six different tracks and may reveal more than one-half milli-inch of variation between the inner and outer track centers.

Using the DDD

Dysan Corporation has been making the DDD for about two years. It was conceived as a way for manufacturers to verify the operation of completed systems quickly, without having to reach any interior test points. Field service personnel have been using it to diagnose drive problems at the customer's location. Recently, the company has been promoting the DDD to end users at various computer shows.

To assist the end user in creating a program that reads the DDD and displays the results of the various tests, the Customer Engineering Division of Dysan Corporation developed the original (version 1.0) DDD program. Designed to run on their "Single Card Computer" with a Hazeltine 1500 terminal, version 1.0 employs direct-memory-access (dma) transfers of read data and requires an 8080 macro assembler for assembly. Luckily for those of us with other types of equipment, constants are clearly defined at the beginning of the code, and the hardware-dependent routines are collected at the end of the code. I found the macro facility was rarely used, so I expanded the few examples into code that ASM can handle. Since my terminal does not have a clear-to-end-of-line function, I inserted a conditional that either prints a string of blanks when it is absolutely essential or allows those who have the function to benefit from its greater speed.

This still leaves a few basic requirements: The program depends heavily on being able to clear your terminal screen and address your cursor. If you have a really dumb terminal, you will have to completely rethink the graphic display parts of the code. The program, as written, commands a Western Digital 1793 floppy disk controller chip; if your system does not use this chip, you will have to understand both the 1793 and your own controller well enough to translate commands between them. Certain controller chips are not capable of performing all the commands the program uses, so some ingenuity may be required. Different systems have different ways of selecting which drive is to be tested. I have added to version 1.0 a routine to select any of four drives connected to my CCS 2422 controller. You'll have to understand how your system goes about selecting drives. Finally, as should be obvious, you'll need to be able to deal with 8080 assembly language in order to get the DDD program working.

Nothing inherent in the DDD limits it to CP/M or 8080 systems. It is available in models for 8-inch or 5¼-inch drives; single or double sided; single or double density; 48, 96, or 100 spi. Since the DDD program has been placed in the public domain, it may be used as an example to construct versions for other machines, processors, and operating systems. If you develop a version for a popular computer, I hope you'll make it available on a bulletin board or software exchange.

If you'd like to take the mystery out of your flexible disk system, and you're willing to be guided through some hardware-intensive assembly language, here's your agenda. First, order a DDD. If you have double-sided drives, you'll need the double-sided DDD in order to test both heads. If all your drives have double-density capability, you'll want to test them in this more stringent mode. If some are single density only and your doubles can read single, perhaps you'd

prefer a single DDD for all of them. You'll need to specify the disk size and number of tracks per inch. Call your local Dysan representative, or call the factory at (408) 988-3472 or (800) 551-9000 outside Northern California, and be prepared to spend $30.00 for a single-sided DDD or $40.00 for a double-sided DDD, plus shipping and applicable taxes. For more information on DDD applications, call Dysan's Tech-Line at (408) 734-1624.

Next, you'll need machine-readable source code for the program that reads the DDD and displays the results. DDD11.ASM (or DDD11.AQM in the "squeezed" version) may be available on your local RCPM system. If you have a phone line, modem, XMODEM-Protocol transfer utility, and "Unsqueeze" if DDD11.AQM is used there, you can copy the program yourself for free. If you must call long distance, be warned that DDD11.ASM is a 47K file. You might be better off downloading it from Micronet's CP/M Interest Group "Access area 1." If you prefer to purchase the code on disk, contact your favorite source for CP/MUG disks or write CP/MUG, 1651 Third Avenue, New York, NY 10028. Should all

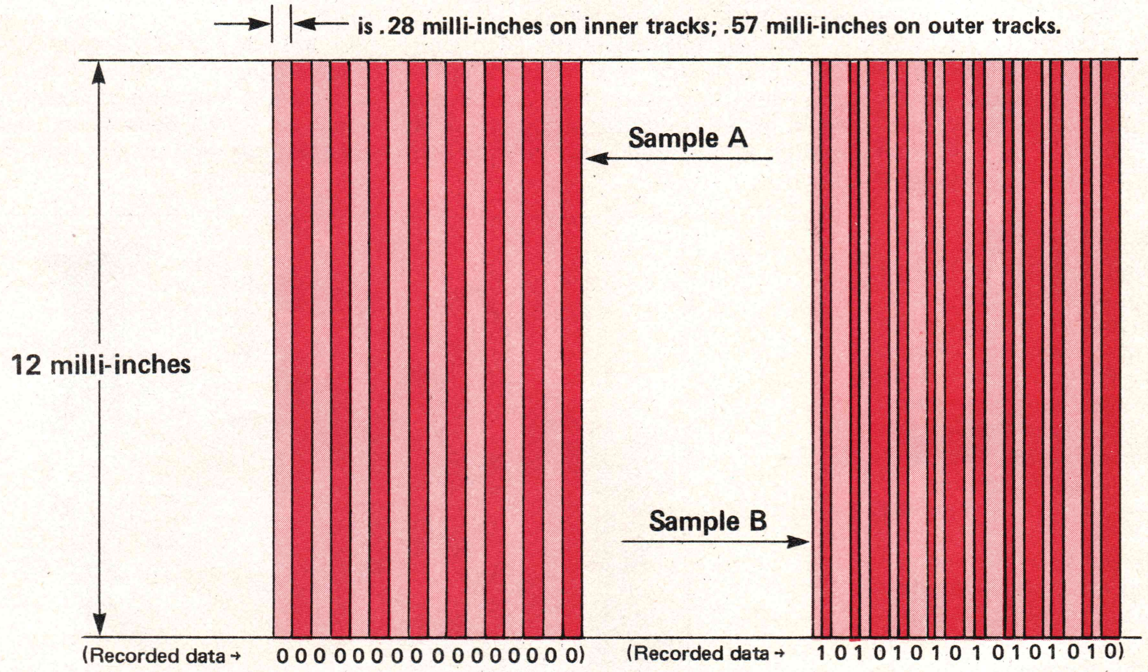


Figure 1.

The actual magnetic pattern on an 8-inch diskette, to scale.

The black lines are *not* on the diskette. There are no prearranged "boxes" to be filled in, only uniform magnetic particles. When you write on the blank diskette, you arrange these particles in alternating stripes of "north" and "south" poles (the light and dark areas in the figure). It is not the polarity itself but the *change* in polarity that is significant. Sample A is a two-byte string of "0" bits, which means that there are no (zero) transitions with a bit space. Sample B alternates between "1" and "0" bits; a transition within a bit space is read as a "1."

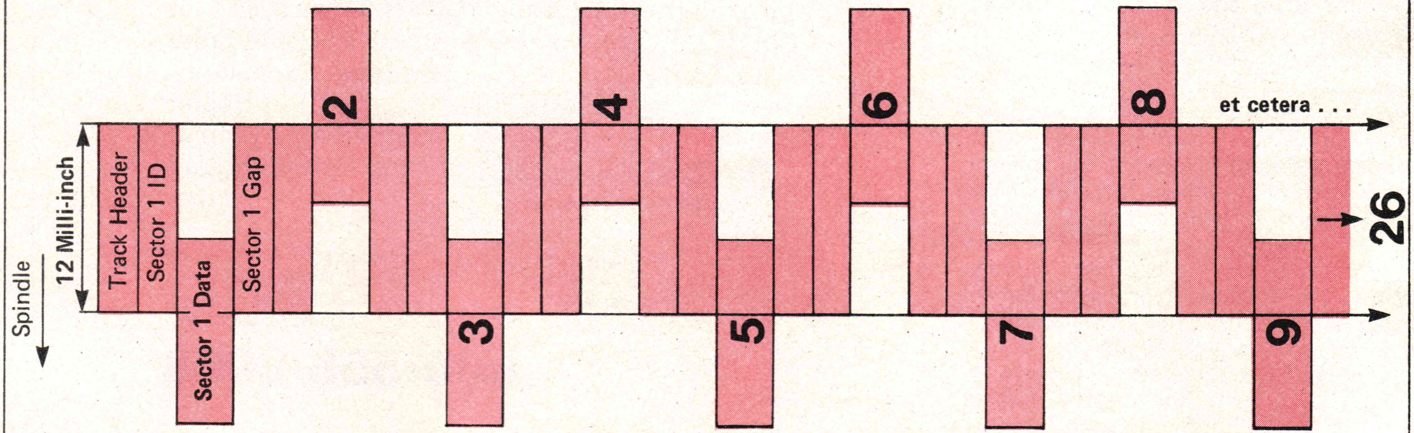


Figure 2.

An alternative offset track.

Each data sector is 128 bytes, while sector IDs are 31 bytes and sector gaps are 29 bytes or less. The track header is 73 bytes. The circular track is shown here stretched out straight for convenience — tracks form complete circles on diskettes.

Don't call her cheap. Call her beautiful.

The Bonnie Blue™

Word Processing System for the IBM Personal Computer

It's obvious what makes her cheap. But what makes Bonnie Blue so beautiful? Bonnie Blue is a totally new word processing system for the IBM Personal Computer. Bonnie Blue really gives you the combination of power and convenience. Ignore the price for a minute, and listen to this.

The Full System

Bonnie Blue is a single program, with three modules, organized as overlays. The heart of the system is the full-screen editor. As we'll show you, it's worth more than the price of the whole system. But the Main Menu also provides access to a complete print formatting module, and the Toolbox. Read on.

Full Screen Editor

Bonnie Blue has a full screen editor, and a complete set of cursor control operations lets you move anywhere on the screen, or scroll the text up or down. All cursor and edit controls are single keys, so there are no sequences to learn. Bonnie Blue can move, copy, delete and paste blocks of text. It can also insert a file into your text, or extract a block of text onto a file. You can find strings and insert, delete or replace them, either individually or over a range of lines.

Bonnie Blue does not force you to go back and forth between different "modes"; creating and editing text are part of the same process. You can turn on or off automatic wraparound, centering, justification, and addition of underline or other attributes to the text. By turning these off, you quickly turn Bonnie Blue from an excellent word processor into a superb text editor, with special features for programmers, such as auto indent.

You can remap any key to any other key; this is done within the program, and saves you the cost of the whole package by itself. You can put any displayable character into your text, with lines up to 220 columns wide.

Bonnie Blue makes unique use of the display attributes (underline, boldface, etc.). You can "paint" these attributes onto your text, and then, when printed, they can be mapped to any single feature or combination of features which your printer can provide. For example, you might want text which is underlined on the screen to be underlined when printed, but you could just as easily make it underlined italics, or reverse video might mean boldface and wide text. The dot command .USAGE can change what the attributes mean during the printing process. All files produced by Bonnie Blue are DOS standard.

Print Formatter

The print formatting module is coordinated with the editor.

Our first print module supports all the features of the Epson MX printers with Graftrax Plus. By the time this ad appears, we will be supporting other popular dot matrix and letter quality printers as well.

The print module can print multiple copies, controlled from a "mailing list" file, with insertion of data from the file into the document, for the production of form letters.

The Tool Box

Bonnie Blue has a full set of "software tools" which bring Unix-type productivity to text manipulation. There are tools to include and concatenate files, count words, sort, and transliterate characters.

We plan continued expansion of the Tool Box including a list manager, telecommunications, and aids for the C programmer, including our version of lint.

Can There Be More?

Bonnie Blue is not copy protected, so it's great for a hard disk environment. The manual is large and thorough. It runs incredibly fast. And there are many other features too numerous to mention here.

If you have an IBM PC, 128k memory, and an Epson or Epson-compatible printer, order today. Or send for our brochure. We are developing versions for other 8086 and 80186 based systems.

Yes. Send me the Bonnie Blue Word Processing System. Enclosed is my payment of $50 (NY State residents add 7% sales tax). Mail coupon and payment to:
Bonnie Blue Software, P.O. Box 536, Liverpool, NY 13088

☐ Check enclosed ☐ VISA ☐ MasterCard Sorry, no COD.

Credit Card No.______________ Expires______

Signature______________

Name______________

Address______________

City______________ State______ Zip______

Company______________

☐ Please send literature. I have a ______________

Only $50

IBM Personal Computer is a trademark of IBM Corp.
Epson is a trademark of Epson America, Inc.

Circle no. 11 on reader service card.

CA 95466-0024, and enclose $10.00. Listing One (page 48) contains a slightly abbreviated version of DDD11.ASM, called DDDCompact.ASM.

Finally, you'll need some information about your hardware. Check your terminal's manual to see if it gives clear-screen and addressable cursor-control commands. See if your disk controller manual gives control port addresses, command formats, status codes, and flow charts of command execution. If the information isn't there, ask the manufacturer if more detailed programming information is available. While a source listing of your CP/M BIOS would probably reveal what you'll need to know about the disk controller, this will be more difficult to decipher. If your hardware is based on an LSI controller chip, the chip manufacturer's data sheets will be very helpful. Save yourself frustration by collecting your hardware information before you begin – while you wait for the DDD disk and DDD11.ASM to arrive.

Customizing DDD11.ASM for Your System

Once you've chosen which DDD format you need, collected your hardware manuals, and are seated at your keyboard with DDD11.ASM in your editor, we can begin. You've read the headers down to "Program Source Code." The first instruction to the assembler is "wboot equ 0" – this is nearly always true. If you happen to be dealing with a nonstandard CP/M system, you probably are well aware of the fact; you've needed special versions of programs from day one with your system and will have to modify my recommendations as we proceed.

Next come the "Conditionals." Here you must decide how committed you are to good programming style. The quick way to customize a program is to step through the code and change only a few magic numbers until it works. The right way, which will pay off handsomely if you should want to give the program to a friend or need to understand what you did come next year, is to create conditional assembly options here for both your terminal and your disk controller. Whenever you need to alter a bit of code in the body of the program, insert "if My____," enter your changes, and then write "endif." By setting "My____" equ "true" in the "Conditionals" section, you insert all your changes into the assembled code. If your friend wants the original code back, he or she can remove all your changes by setting "My____" equ "false." Unless you happen to have one of the existing conditional systems, set these conditionals equ "false," add similar lines for your terminal and controller, and set yours equ "true." Set "lineclr" true or false according to whether your terminal can clear-to-end-of-line with a single command.

The "Macro Call" section has been commented out, and all examples of its use have been expanded by hand, so that you can use a simple assembler like ASM.

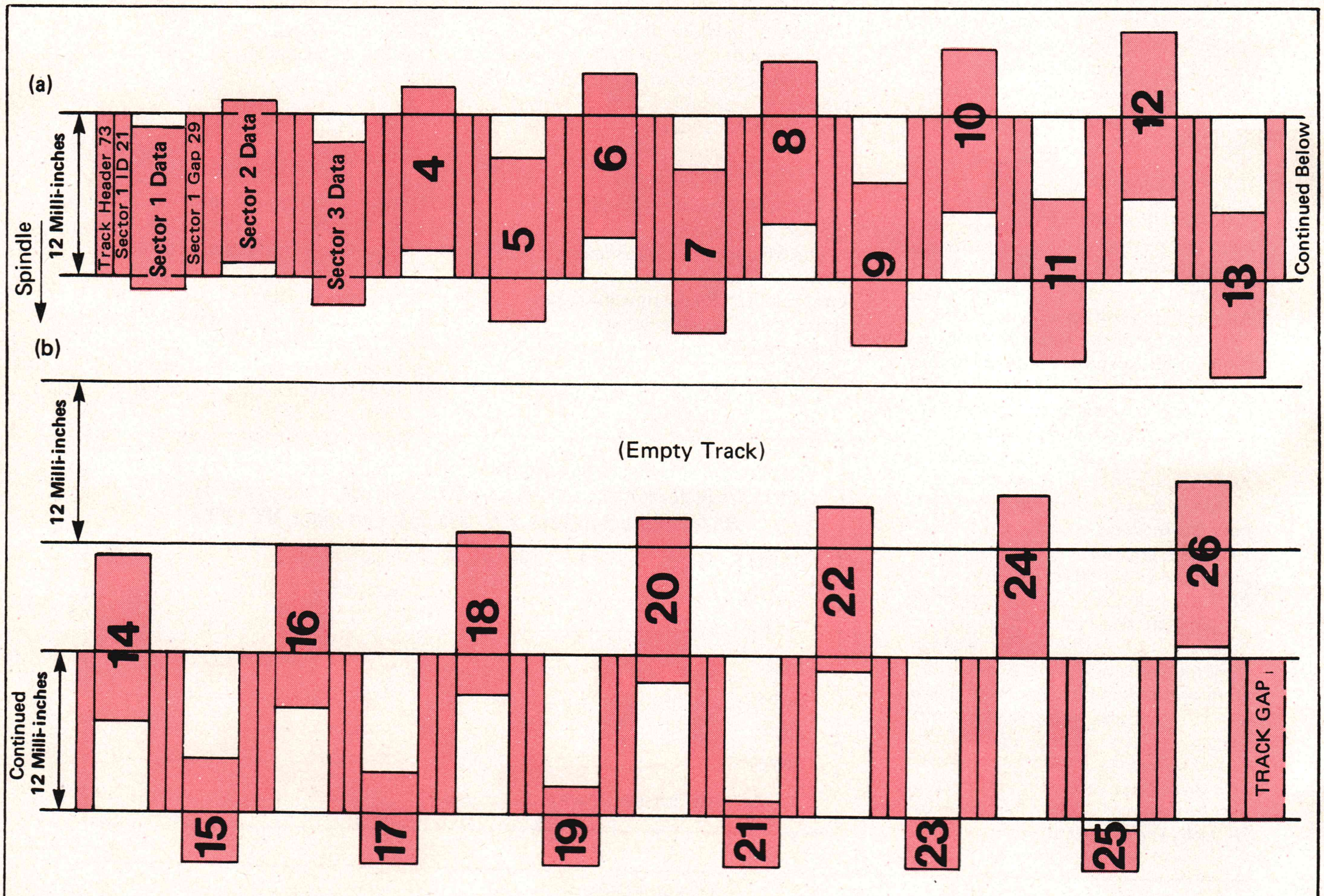
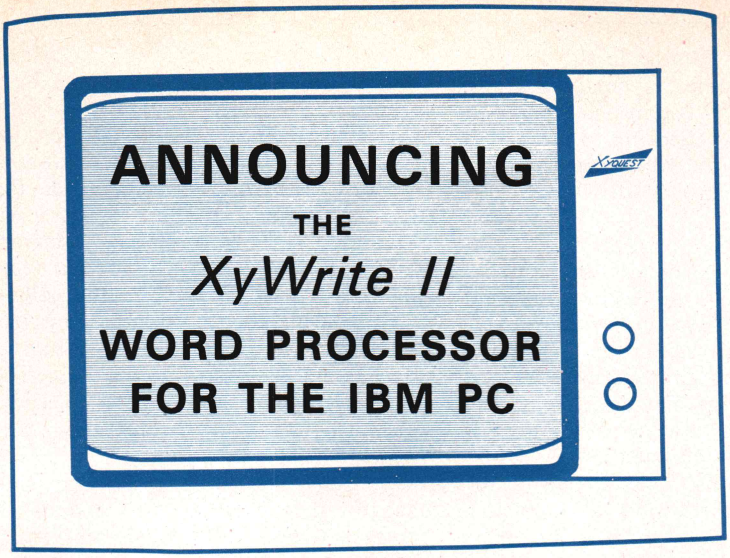


Figure 3.

A progressive offset track.

Vertical scale is 75 to 150 times horizontal, depending on the track number. The entire track is displayed here, split into two parts: sectors 14 through 26 (at (b), the bottom part of the figure) are displayed with a neighboring empty track in order to show how the offset sectors begin to bleed out into the adjacent empty tracks; sectors 1 through 13, at (a), are shown without neighboring tracks.



from XYQUEST

We hereby make you an UNBELIEVABLE OFFER:

For only $7.00 ($5.00 for the disk, and $2.00 for handling), we will send to you a DEMO DISK of XyWrite II, THE ONLY WORD PROCESSOR to take advantage of the SPEED and POWER of the IBM PC.

We believe in our program so we want to make it easy for you to try it. We want you to compare XyWrite II against the rest. You will quickly see the lightning responsiveness of the program: it'll take your breath away.

FASTEST PROGRAM ON THE MARKET

SIMULTANEOUS MULTIPLE FILE ACCESS

SPLIT SCREEN WRITING (HORIZONTAL OR VERTICAL)

SPLIT SCREEN EDITING (MOVE TEXT BACK & FORTH)

HORIZONTAL SCROLLING

EDIT OR WRITE WHILE PRINTING

INTEGRAL MATH PACKAGE

COLUMNAR POSITIONING

INDEXING, FOOTNOTES, HEADERS/FOOTERS

and MUCH, MUCH MORE

Find out what real speed power is on your IBM PC. Compare XyWrite II with WordStar, EasyWriter 1.1 and II, Benchmark 3.0, PowerText, Volkswriter, FinalWord, Select, WordPlus, PerfectWriter - or any of the others.

You have nothing to lose. The high quality diskette is yours to keep. When you decide to buy XyWrite II, you receive a $7.00 discount off the $195.00 list price.
Call or write us today at (617) 275-4439. Ask for John.

WordStar is a registered trademark of MicroPro International. Benchmark is a registered trademark of Metasoft Corp. PowerText is a registered trademark of Beaman Porter, Inc. Volkswriter is a registered trademark of Lifetree Software, Inc. FinalWord is a registered trademark of Mark of the Unicorn. Select is a registered trademark of Select Information Systems. WordPlus, is a registered trademark of Professional Software, Inc. PerfectWriter is a registered trademark of Perfect Software. EasyWriter is a registered trademark of Information Unlimited Software, Inc. IBM and IBM PC are registered trademarks of International Business Machines Corporation.

XyQuest Inc.
P.O. Box 372
Bedford, MA 01730

Circle **no. 130** on reader service card.

Ignore it. Under "Console Functions" comes your first chance to say "if Myterminal." Set as many of the variables as your terminal uses to the values given in your manual. Don't forget the "endif." Check the ASCII code equates, especially backspace, for compatibility with your terminal.

In "Floppy Controller Port Equates," you tell the program where to send commands for the disk controller. Set up a conditional for your controller; you can probably copy these addresses directly from your BIOS source, if you have it. Note how the base address is given in absolute numbers, and all the other addresses are given as offsets from the base. Most systems with a 1793 controller chip will have these same offsets from whatever base address the system uses.

In the "if Dysan" conditional, "dma" is a port address where commands to the direct memory access controller are sent. Each time this controller is asked to read the disk, it is sent an address in memory at which to begin storing what is read. It reads the disk and puts the data directly into memory, while the CPU does other tasks, by using the slivers of time between CPU memory accesses. In my CCS conditional, "dma" is a memory address where a pointer to a memory buffer is stored. In this non-dma or programmed I/O transfer system, after telling the controller to read the disk, the CPU must be ready to grab each byte as it is read and to move it into memory before the next byte is ready. The CPU looks at "dma" to find out where in memory to begin storing the read data. Obviously, the CPU cannot do anything else while it is busy moving in data; in fact, to keep up with a double-density 8-inch disk it must run at least at 4 MHz.

"Floppy Controller Commands" is another section you might copy from your BIOS source. Command "sdma" tells the Dysan controller to begin transferring data; if you have a dma-type controller, it should have a similar command. Variable "fdelay" is used in the "delay" routine, where time is being carefully counted. Instead of issuing a command to the controller and checking to see if it is processed, this routine dispenses the delay time in known chunks of 50 microseconds. If you don't have a 4 MHz Z80, and you want the timing routines to work, "fdelay" and the "delay" routine will have to be juggled until the loop takes 50 microseconds. "Indxbyt," "busybyt," and "drqbyt" are masks that select the appropriate bit from the status register of the controller chip. For example, if the status register is read, and the logical "and" function is done with "busybyt," a 0 will result unless the bit signifying "busy" is set high. A bit-by-bit explanation of status register contents should be available in your controller chip data sheet. The other variables in this section go directly to the controller chip and can be determined from the manufacturer's data sheet, if not from your BIOS source.

In "DDD Diskette Variables," the total number of tracks and the number of sectors per track in your chosen format are set. Note that several other parameters are defined in terms of sectors and need not be changed. "Ref" is a magic number that will be discussed later under "Index Timing." Ignore it for now. Those of you with nonstandard CP/M systems will have to change the "org" address. Otherwise we're already down to the actual program!

Step on down to "Clear Console Display" – the code in between should run unchanged on any CP/M system. If your terminal, like the Hazeltine, needs an extra lead-in byte ahead of the actual clear-screen command, you'll need to put in an "if Myterminal" here. "Clear Display Line" has two conditionals: one for terminals with a clear-to-end-of-line command, which is active if "lineclr equ true," and another that prints a string of blanks for terminals without this feature. If you've set "lineclr" true, and if your terminal needs a lead-in byte or other special treatment, insert an "if Myterminal AND lineclr" as I have for the Hazeltine. Again, in "Print String of Text," insert a conditional under "text1" for lead-in type terminals. If your terminal should need the row position sent before the column position, accomplish that here by incrementing the HL register pair twice after the position cursor command (to get the row position) and then decrementing HL to get back to the column position. This code section assumes that the upper left-hand corner of your screen is called 0,0. If your terminal places 0,0 somewhere else, you'll have to add a constant bias to the coordinates before you send them.

"Index Timing" may require major changes. In the Dysan dma system, the command "read address" is sent to the dma controller, "start dma" is sent, and the CPU is set to counting the time until the controller is finished doing its direct memory access. If you have a dma-type controller, you'll use this scheme as it is. If you have a programmed I/O controller, the CPU cannot transfer data and count time simultaneously. My solution, in the CCS conditional, is to send the "read address" command and immediately set the CPU to counting time until "drq" comes true. When the first byte of address data is ready to be transferred to memory, the disk controller sets "drq" true, telling the CPU to "come and get it." The CPU immediately determines (lhld dma) where to put the byte, reads it from the data port, and moves it to memory. It increments the memory address and goes after the next byte. "Read address" reads only six bytes of data, so I wrote six reads inline.

Since the non-dma CPU only counts time before the data transfer, the time counting loop after that code is for the Dysan controller only. But because the Dysan dma controller counts both the time from the "read address" command to the first data byte and the time needed to transfer six bytes of data, it will get a longer time count. This is where "ref"

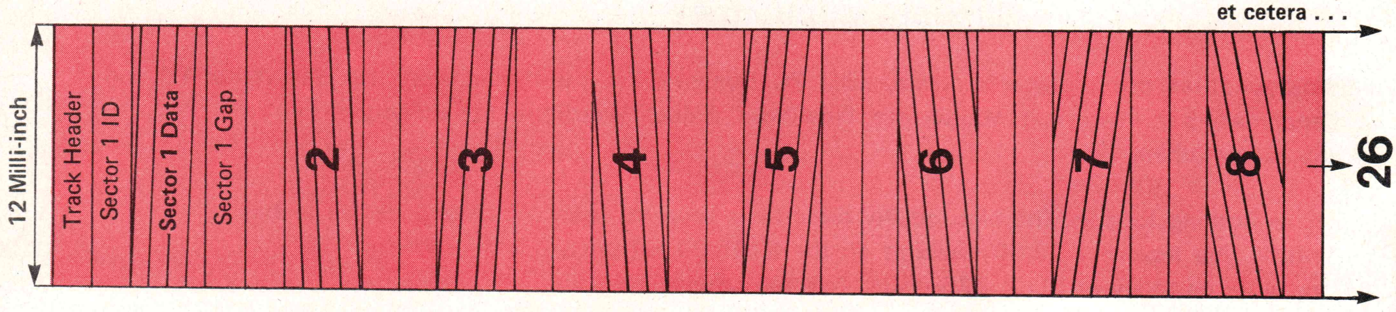


Figure 4.

An azimuth rotation track.

Data sectors, sector IDs, sector gaps and track headers take the same number of bytes as in Figure 2. Azimuth rotation is *highly exaggerated!* Actual rotation is from 18 to 42 minutes – less than a degree. Again, the track is drawn straight for convenience.

comes in. "Ref" is a magic number or, more bluntly, a fudge factor, which makes the final count read however we want it. Since "ref" is subtracted from the total time count, the non-dma controller will require a smaller "ref" value than the dma controller. Dysan's value is experimentally set with their Analog Alignment Diskette, whose track has a standardized 200 microsecond time from index hole to address mark. My value of 24EH is estimated to make my DDD also give about 200 microseconds on my drives – hardly a precise method, but a relative measure is better than no measure at all.

"Index Timing" is the first section to introduce the problem of time counting. You've undoubtedly noticed that many of the instructions are followed by numbers in the comment field. These numbers give the execution times in microseconds for those instructions on a 4 MHz Z80. If your CPU is clocked at 2 MHz, the instructions will take twice as long. But disk reads of data happen at the speed of the moving diskette, regardless of the CPU clock. If you intend to properly adapt this routine to a different clock rate, you will have to manipulate not only the timings here, but also the delay provided by "delay" under "FDC Delay to Process Commands" and the loop count "fdelay" in "Floppy Controller Commands." Unless you enjoy mathematical puzzles, I suggest you join me in a cut-and-try fudge. In "Index to Index Timing" and its "RPM Timing Loop," no disk reads are involved, only time counting between index pulses. Here a 2 MHz processor should get exactly half the proper time, and a mathematical correction is reasonably simple. Translate the instruction times into their corresponding values at your clock speed, and set fewer loop passes or adjust the trailing "inx d" instructions to give the required 100 microsecond loop.

"Drive Select" is your chance to be original. The version 1.0 option simply selects one drive only. My CCS option can select one of four drives connected to a CCS 2422 controller, but it does not deal with double-sided drives. The code down to "jmp select" should be pretty universal. After that, you'll have to determine from your manual (or crib from your BIOS source) the codes needed to select drives under your controller and then invent a way to turn "A,B,C, or D" into whatever codes you need. If you need to select sides on a double-sided drive, duplicate the top part of the CCS option then display a "Which Side?" message, translate the response into the proper code, and send it to the proper port.

In the "Read Sector Routine," we again have to separate dma controllers from non-dma controllers. With a dma controller, just issue the go command and

check to see when the status register says all the data are transferred. With a programmed I/O controller, you'll have to tell the CPU every step to take to get the data into memory. Under "read1" in the CCS option, I loaded HL with the memory destination of the first byte and initialized a loop counter. Each loop through the data transfer code reads four bytes, so to read an 8-inch single-density sector takes 32 (20H) loops. Double-density 8-inch sectors require 64 (40H) loops. This routine can probably be adapted from the disk read routine in your BIOS. You could call the BIOS routine, but that would make your DDD program nonportable.

"Program Disk Controller" originally output a series of commands to a dma controller, reading them in serial form from the table that followed. If you have a dma controller that needs to be "primed" with a string of commands, create your own conditional table and adjust "Bytes count" for the number of bytes you need to send. Your manual should explain the procedure; otherwise, find a data sheet on the actual chip used or copy from your BIOS. Even a non-dma controller needs some priming. Mine will look at memory address "dma" (remember "Floppy Controller Port Equates"?) and expect to find the address of "secbuf," a memory buffer to move data to. The other item needed to prime it for a disk read is the setting of "autowait," a function of the CCS 2422 controller card that puts my CPU into a wait state from the time it sends a disk read command until the first byte of data is ready. The catch is that to set "autowait" I have to write to the same port that does disk select; thus I save the bits I used to select the current disk and sent them here with a double function.

"Delay" has already been dealt with (or fudged), so unless you'd like to customize some of the console displays or messages, we've reached the end of the source code! Run it through your assembler and clean up any trivial errors, and the program should be yours. To make it a useful tool, you'll need to create a "before" picture of your drives' operation. Run all the tests on all your drives and write down the results. There are no "correct" results for these tests; however, you'd like to be able to read as many sectors as possible and to have your readings symmetrical around the center of a properly aligned track. If you record a baseline while your drives are working, you'll be able to identify any subsequent changes as possible causes of new malfunctions.

Finally, be aware that flexible diskettes, including the DDD, change size with temperature and humidity. A change of 18°F can move the outer tracks of an 8-inch disk up to 1 milli-inch. On an extremely humid day, the outer tracks may "grow" another 3 milli-inches. If your drives are in a poorly ventilated enclosure where the temperature rises to 125°F, that alone can move traks 7 milli-inches out of alignment. Run your baseline tests under normal conditions of operation after your system is well warmed up, and be sure the DDD has had time to acclimate to the room and to the temperature inside the drive you're testing. Proper drive alignment once the drives are packed into your enclosure may be considerably different from alignment in the open on a test bench.

DDJ

(Listing begins below)

Diagnostic Diskette Listing

(Text begins on page 40)

```
;       DIGITAL DIAGNOSTIC DISKETTE tm  APPLICATION PROGRAM
;       BY LOREN AMELANG, BOX 24, PHILO, CA      95466 0024
;       THIS CODE EXCERPTED FROM VERSION 1.1    (DDD11.ASM)
;       (THE EXCERPT DOES NOT INCLUDE STEPPER HYSTERESIS
;                               OR INDEX TIMING CHECKS...)
;       This document contains information developed by
;       Dysan Corporation (Dysan) and is furnished for
;       information only.
;       Dysan makes no warranty or representation (expressed
;       or implied) with respect to the accuracy, completeness
;       or usefulness of the information contained herein.
;       Further Dysan assumes no responsibility for liability
;       or damage of any kind which may result from the use
;       of the information contained herein.
;       THIS PROGRAM IS IN THE PUBLIC DOMAIN, AND MAY BE
;                     FREELY COPIED AND DISTRIBUTED
;
;
;       *************************************************************
;
;       USAGE:
;
;       To use this program, a digital alignment disk is required.
;       You may purchase the required "Digital Diagnostic Diskette" (tm)
;       directly from Dysan Corporation.   Available DDD(tm) models are:
;
;       SIDES:...........SINGLE SIDED............DOUBLE SIDED
```

(Continued on page 50)

8087 Number Cruncher

for CompuPro,® NEC and other microcomputers.*

*North Star and Zenith versions by 4th quarter, 1983.

"This will be a boon to those of you who wish to take advantage of **8087** for fast floating-point math but don't want to trash your present CPU board . . ." —Ray Duncan, ***DR. DOBB'S JOURNAL***

". . . the result (from the **Hudson 8087 Support Board**) is well worthwhile." —Jerry Pournelle, ***BYTE***

Perform math operations at the speed of a mainframe computer. The **Hudson 8087 Support Board**™ plugs right on your processor board, permitting you to perform numeric operations **UP TO 300 TIMES FASTER!** Installation takes less than five minutes—requires no software modification.

Additionally, the **8087** is being supported by virtually every major language house, including Digital Research and Microsoft.

NOW 8-BIT LANGUAGE SUPPORT FOR THE 8087 SUPPORT BOARD

Typical double precision (64-bit) **benchmarks:****

Operation (5000 iterations)	**FORTRAN-80**	**FORTRAN-87/BASIC-87** (5 MHz)	**FORTRAN-87/BASIC-87** (8 MHz)
multiplication	32 sec.	2.4 sec.	1.5 sec.
division	62	2.5	1.6
sine or cosine	380	3.1	2.0
logarithm	390	2.6	1.7
square root	500	1.7	1.1

FORTRAN-87™ and BASIC-87™ put the **8087** to work for you using Microsoft's popular BASIC and FORTRAN compilers. Use your regular CP/M 2.2® and FORTRAN-80/BASIC-80 compilers on your CompuPro, Zenith, or North Star dual-processor systems interfaced with FORTRAN-87/BASIC-87 and utilize the floating point power of the **8087 SUPPORT BOARD**, increasing your math processing up to several hundred fold.

8087 SUPPORT BOARD: $495.00

(Includes 5 MHz 8087 processor; 8 MHz 8087 $95 extra)

FORTRAN-87: $200.00 BASIC-87: $200.00

Postage and handling included. Prepaid or C.O.D. orders accepted. Give street address for UPS delivery, and allow 3-6 weeks. Dealer inquiries welcomed. Specifications subject to change without notice.

**Results were obtained on a CompuPro 8/16 computer with an 8085 running at 6 MHz and an 8088 running at 5 MHz and 8 MHz respectively.

Hudson Marketing

829 Rosewood Court, Walnut Creek, CA 94596 — **(415) 944-9680**

Circle **no. 48** on reader service card.

Diagnostic Diskette Listing (Listing continued, text begins on page 40)

```
;       DENSITY:......SINGLE(FM) DOUBLE(MFM)...SINGLE(FM) DOUBLE(MFM)
;       SIZE & TPI:
;       8"      48 TPI   808-100   808-200       808-300   808-400
;       5.25"  48 TPI    508-100   508-200       508-300   508-400
;       5.25"  96 TPI    506-100   506-200       506-300   506-400
;       5.25" 100 TPI    CALL FOR AVAILABILITY    CALL FOR AVAILABILITY
;
;
;       PRICES:  Single Sided - $30.00     Double Sided - $40.00
;                Plus appropriate sales taxes and shipping costs
;                Prices and models subject to change without notice
;
;
;       ORDERING:  Contact your local Dysan sales Rep. or call
;                  Inside Northern California:  (408) 988-3472
;                  Outside Northern California: (800) 551-9000
;
;
;       TECHNICAL INFORMATION:  For information regarding DDD(tm)
;                               applications contact Dysan's
;                               CE Division TECH-LINE: (408)734-1624
;
;
;       ***************************************************************
;
;       PROGRAM SOURCE CODE:
;

wboot   equ     0                  ; warm boot address

bdos    equ     5                  ; jump address for bdos calls


*********************
*** "CONDITIONALS" ***
*********************
;
;       Choose the appropriate assemble time options for your system,
;       or use this space to add new system options.
;
true    equ     0ffffh
false   equ     not true
;
Dysan   equ     false   ;set this true for Dysan controller
CCS     equ     true    ;set this true for CCS 2422 controller
                        ;       programmed i/o transfer, etc.
lineclr equ     false   ;set this true if your console has
                        ;       a clear-to-end-of-line command
Hazel   equ     false   ;set this true for Hazeltine 1500 terminal
                        ;       (requiring "lead-in" to commands)
Loren   equ     true    ;set this true for Loren's homebrew terminal...
;
;ADD CONDITIONAL OPTIONS FOR YOUR ENVIRONMENT HERE...
;
;


**************************
*** "CONSOLE FUNCTIONS" ***
**************************

;       The following are direct console commands
;       which can be altered for most consoles.

;       note:
;            The text print routine must be altered
;            if "Row" is sent before "Column" or
;            bias is needed.

```

```
         if      Hazel                 ;version 1.0
add$cur  equ     17                    ; address cursor command
clr$co   equ     28                    ; clear to foreground spaces
lead$in  equ     126                   ; console lead-in
end$ln   equ     15                    ; clear to end of line
         endif

         if      Loren                 ;version 1.1
add$cur  equ     01                    ; address cursor command
clr$co   equ     05                    ; clear to foreground spaces
         endif

cr       equ     0dh                   ; carriage return
lf       equ     0ah                   ; line feed
esc      equ     1Bh                   ; escape
bs       equ     8                     ; back space
bell     equ     7                     ; audio alert


****************************************
*** "FLOPPY CONTROLLER PORT EQUATES" ***
****************************************

;        Change the following Port assignments for
;        Your system.
;
;        "FDC"  Acronym  Floppy Disk Controller
;
;        FDC port assignments:
;
         if      Dysan                 ;version 1.0
fdc      equ     0f8H                  ; Base port address FDC
```

(Continued on next page)

Diagnostic Diskette Listing (Listing continued, text begins on page 40)

```
;
cmd      equ     fdc              ; command reg.
stat     equ     fdc              ; status reg.
trk      equ     fdc+1            ; track reg.
sec      equ     fdc+2            ; sector reg.
data     equ     fdc+3            ; data reg.
dma      equ     fdc+4            ; Dma controller
dsel     equ     fdc+5            ; drive select
         endif

         if      CCS              ;version 1.1
fdc      equ     30H              ; Base port address FDC
;
cmd      equ     fdc              ; command reg.
stat     equ     fdc              ; status reg.
trk      equ     fdc+1            ; track reg.
sec      equ     fdc+2            ; sector reg.
data     equ     fdc+3            ; data reg.
dma      equ     004CH            ;memory address of pointer to
;                                         sector buffer
dsel     equ     fdc+4            ; drive select port:
                                  ;output BX for single density
                                  ;       FX for double density
                                  ;   where X = 1 for drive A
                                  ;             2 for drive B
                                  ;             4 for drive C
                                  ;             8 for drive D
         endif

*************************************
*** "FLOPPY CONTROLLER COMMANDS" ***
*************************************

;        These are common to most systems using the 1793.
;        sdma is a Dysan controller command, ignored in V. 1.1.

clear    equ     0d0h             ; clear FDC
sdma     equ     87h              ; Start Dma Transfer
rsec     equ     80h              ; read sector
raddr    equ     0c0h             ; read address
seek     equ     18h              ; seek track
restore  equ     8                ; home Head(s)
sdelay   equ     4                ; seek delay flag

*************************************
*** "FLOPPY CONTROLLER VARIABLES" ***
*************************************

rate     equ     1                ; step rate
retry    equ     1                ; read retries
fdelay   equ     11        ; FDC delay
                           ; see "delay" routine - this value is for
                           ; 4 Mhz. Z-80: adjust here or in "delay"
                           ; to suit your clock and processor...
indxbyt  equ     2         ; byte in status reg. when index hole
                           ; is passing sensor
busybyt  equ     1         ; byte in status reg. when controller
                           ; is busy
drqbyt   equ     2         ; byte in status reg. when data is
                           ; waiting to be read by the computer

******************************
*** "DDD DISKETTE EQUATES" ***
******************************

;        Adjust to match your diskette size and format.

ttrk     equ     76               ; total number of tracks
sectors  equ     26               ; sectors per track
```

```
                                            ; 26 for 8" DDD, 16 for 5.25" DDD
lneg        equ      sectors+2                   ; last neg. sector +2
lpos        equ      sectors+1                   ; last pos. sector +2
tsec        equ      (sectors/2)-1          ; centering test
            if       Dysan
ref         equ      798                    ; reference time (single Den)
                                            ; (used in "index 7")
            endif
            if       CCS
ref         equ      24eh                   ; experimentally determined...
            endif
***************************
*** "MAIN PROGRAM LOOP" ***
***************************
            org      100h
;           ====     ====
            lxi      sp,stack               ; init stack
            call     select                 ; select drive
;
main:
            lxi      sp,stack               ; reset stack
            call     cls                    ; clear display
            lxi      h,me1                  ; display Menu..
            call     text
            call     ci                     ; selection?
```

(Continued on next page)

Diagnostic Diskette Listing

(Listing continued, text begins on page 40)

```
                cpi     'R'             ; "Radial"
                jz      radial
                cpi     'A'             ; "Azimuth"
                jz      azimuth
                cpi     'C'             ; "Centering"
                jz      center
                cpi     'H'             ; "Hysteresis"
;               jz      hyster
                cpi     'S'             ; "RPM"
                jz      rpm
                cpi     'I'             ; "Index"
;               jz      index
                cpi     'D'             ;"Drive Select"
                jnz     ex
                call    select
                jmp     main            ;new drive selected - get option

ex:             cpi     'E'             ; "Exit to DOS"
                jz      exit
                mvi     a,bell          ; Not valid
                call    co
                jmp     main            ; try again...


**********************
*** " EXIT TO DOS" ***
**********************

exit:
                call    cls             ; clear display
                jmp     wboot

*************************
*** "CONSOLE  STATUS" ***
*************************

;               returns with zero status flag set.

cstat:
                push    h               ; save all registers
                push    d
                push    b
                mvi     c,11            ; BDOS console status
                call    bdos            ; function call added for V1.1
                pop     b
                pop     d
                pop     h
                ora     a
                ret

**********************
*** "CONSOLE INPUT" ***
**********************
;
;               Returns character in "A" reg.
ci:
                push    h               ; Save all registers
                push    d
                push    b
                mvi     c,1             ; BDOS console input
                call    bdos            ; function call added for V1.1
                pop     b
                pop     d
                pop     h
                cpi     esc             ; Exit?
                jz      main            ; YES!
                cpi     'a'             ; convert to upper case
                rc                      ; only...
                cpi     '{'
```

```
        rnc
        ani     '^'+1
        ret


*************************
*** "CONSOLE OUTPUT" ***
*************************
;
;       Enter with character in "A" reg.

co:
        push    psw
        push    h               ; save all registers
        push    d
        push    b
        mov     e,a             ; Pass char in "E" reg for V1.1
        mvi     c,6             ; BDOS direct console i/o
        call    bdos            ; function call added for V1.1
        pop     b
        pop     d
        pop     h
        pop     psw
        ret


*********************************
*** "INPUT TRACK LOCATIONS" ***
*********************************

;       Input track locations for Radial
;       Hysteresis,Centering and Index tests.
```

(Continued on next page)

Diagnostic Diskette Listing

(Listing continued, text begins on page 40)

```
;       "B" contents Upper Limit upon entry.

inp$trk:
        call    cls             ; clear display...
        call    text
;
inp$trk1:
        call    ci              ; waiting...
        cmp     b               ; within limits?
        jnc     inp$trk2        ; no...
        sui     'A'
        jc      inp$trk2        ; try again
        lhld    temp            ; table pointer
        mvi     d,0
        mov     e,a
        dad     d               ; add offset
        mov     a,m             ; load track
        jmp     track           ; seek to track
;
inp$trk2:
        mvi     a,bell          ; alert
        call    co
        jmp     inp$trk1        ; try again...

**************************************************
*** "RADIAL AND HYSTERESIS TRACK LOCATIONS" ***
**************************************************

rh$tbl
```

```
        db      0
        db      3
;       Note:  some Dysan spec sheets list track 5 rather
;       than 3 as an index track - my Revision A DDD definitely
;       uses track 3...    LA.
        db      38
        db      41
        db      70
        db      73

*************************
*** "INDEX TRACK TABLE" ***
*************************

ix$tbl:
        db      0
        db      76

***********************
** "CENTERING TRACKS" ***
***********************

cen$tbl:
        db      35
        db      44
        db      47


****************************
*** "PRINT DECIMAL NUMBERS" ***
****************************
;
```

(Continued on next page)

Diagnostic Diskette Listing

(Listing continued, text begins on page 40)

```
;       Enter with number to be printed in
;       'DE' reg.

pnum:
        push    h               ; Setup Stack
        lxi     h,10            ; Terminator...
        push    h
        push    h
;
pn1:
        call    div16           ; Divide Number By (10)
        mov     a,d             ; Check result
        ora     e               ; = 0
        jz      pn2
        xthl
        dcr     l
        push    h
        lxi     h,10
        jmp     pn1             ; Divide Again
;
pn2:
        pop     d               ; Display Values
        mov     e,l
;
pn3:
        mov     a,e             ; Check for Terminator (10)
        cpi     10
        pop     d               ; Next digit
        rz
        adi     '0'             ; Add ascii Bias
        call    co
        jmp     pn3


*****************************
*** "16 BIT SUBTRACTION" ***
*****************************
;
;       Subtract "DE" from "HL"
subde:
        mov     a,l
        sub     e
        mov     l,a
        mov     a,h
        sbb     d
        mov     h,a
        ret


*******************************
*** "COMPARE "DE" TO "HL" ***
*******************************

dehl:
        mov     a,l             ; compare "E" to "L"
        cmp     e               ; If not zero return
        rnz
        mov     a,h
        cmp     d
        ret



***********************************
*** "16/24 BIT DIVIDE ROUTINE" ***
***********************************

;       Divide "HL" by "DE"
```

```
;
div16:
        xra     a               ; optional 16 bit divide
;
div24:
        sta     msb             ; Normal 24 bit divide
        shld    msb1
        lxi     h,msb2
        mvi     m,24+1
        lxi     b,0
        push    b
div25:
        mov     a,e
        ral
        mov     e,a
        mov     a,d
        ral
        mov     d,a
        lda     msb
        ral
        sta     msb
        dcr     m
        pop     h
        rz
        mvi     a,0
        aci     0
        dad     h
        mov     b,h
        add     l
        lhld    msb1
        sub     l
        mov     c,a
        mov     a,b
        sbb     h
        mov     b,a
        push    b
        jnc     div26
        dad     b
        xthl
;
div26:
        lxi     h,msb2
        cmc
        jmp     div25
*******************************
*** "CLEAR CONSOLE DISLAY" ***
*******************************
;       Clear console to foreground spaces.
cls:
        if      Hazel
        mvi     a,lead$in       ; lead in
        call    co
        endif
        mvi     a,clr$co        ; clear to foreground spaces
        jmp     co
*****************************
*** "CLEAR DISPLAY LINE" ***
*****************************
cline:
        if      lineclr
        mvi     a,0             ; starting column
        sta     column
```

(Continued on next page)

 (Listing continued, text begins on page 40)

```
            lxi     h,set$cur       ; set cursor
            call    text
            endif

            if      Hazel AND lineclr
            mvi     a,lead$in       ; lead in byte
            call    co
            endif

            if      lineclr
            mvi     a,end$ln        ; clear to end of line
            call    co
            endif

            if      NOT lineclr
            mvi     a,11            ; starting column
            sta     column
            lxi     h,set$cur       ; set cursor
            call    text
            lxi     h,blanks        ;address of a string of 56 blanks...
            call    text
            endif

            ret

*************************
*** "FATAL ERROR MSG" ***
*************************

fatal:
            if      lineclr
            call    cline           ; clear line
            endif

            mvi     a,30            ; column position
            sta     column
            lxi     h,set$cur
            call    text
            lxi     h,me2           ; Fatal error msg
            call    text
            ret


********************************
*** "PRINT STRING OF TEXT" ***
********************************

;           Enter with "HL" pointing to text string,
;           terminate string with Null.
;
;           Direction cursor position is used for displays.
;           The value (-1) at the start of string indicates
;           to this routine
;           that column and row position will follow.
;           This format does not necessarily go out to
;           your terminal.   If your terminal needs row sent before column,
;           or a constant bias added (0,0 is not the upper left
;           corner of the screen), you'll have to invent code
;           within this routine to accomplish that.
;
;           This is the internal representation format:
;                 db -1,50,10,'NOW IS THE TIME'
;                     ^  ^  ^
;                     |  |  |_Row position (add bias if needed)
;                     |  |_Column position (add bias if needed)
;                     |_Flag (send console lead-in sequence.

text:
            mov     a,m             ; load char.
```

```
        ora     a               ; end of string?
        rz                      ; yes
        cpi     (-1) AND 0FFH   ; position cursor?
                                ; "and offh" added for ASM compatibility
        jz      text1           ; yes
        inx     h               ; next char.
        call    co              ; output
        jmp     text
;
text1:
        inx     h               ; move passed command

        if      Hazel
        mvi     a,lead$in       ; leading for console
        call    co              ; issue.
        endif

        mvi     a,add$cur       ; address cursor
        call    co
        mov     a,m             ; column position...
        inx     h
        call    co
        mov     a,m             ; row position
        call    co
        inx     h
        jmp     text

*************************
*** "CENTERING CHECK" ***
*************************

;
```

(Continued on next page)

Diagnostic Diskette Listing

(Listing continued, text begins on page 40)

```
;         Centering test requires the FDC to read
;         all the sectors before diskette centering
;         is consider OK.
center:
          lxi       h,cen$tbl       ; track table
          shld      temp
          lxi       h,me9           ; tracks with
          mvi       b,'D'           ; limit
          call      inp$trk         ; seek track
          call      cls             ; clear display
          lxi       h,me10          ; centering msg.
          call      text
          lxi       h,me15          ; frame
          call      text
          lxi       h,me11          ; New track msg
          call      text
;
center1:
          call      rd$trk          ; read centering track
          mvi       a,10            ; set row position
          sta       row
          jc        center3         ; fatal error..
          if        lineclr
          call      cline           ; clear line
          endif
          mvi       a,30            ; set cursor...
          sta       column          ; column position
          lxi       h,set$cur
          call      text
          lda       diff            ; must = 0
          ora       a
          lxi       h,me3           ; "RE-CLAMP DISKETTE"
          jnz       center2
          mov       a,c
          cpi       tsec            ; all sec read?
          jnz       center2
          lxi       h,me4           ; "CENTERING OK"
;
center2:
          call      text
          call      cstat           ; abort?
          jz        center1
          call      ci
          cpi       ' '             ; new track?
          jnz       center1         ; no...
          jmp       center          ; yes..
;
center3:
          call      fatal           ; fatal error
          jmp       center2
**********************************
*** "AZIMUTH ALIGNMENT CHECK" ***
**********************************
azimuth:
          call      cls             ; clear display...
          mvi       a,76            ; seek to azimuth track
          call      track           ; move...
          lxi       h,me12          ; type of test msg
          call      text
          lxi       h,me15          ; frame
          call      text
;
```

```
azimuth1:
        lxi     h,azi$tab       ; set translation table
        shld    xpoint          ; pointers
        shld    ypoint
        call    rd$trk          ; read azimuth track
        mvi     a,10            ; set row position
        sta     row
        jc      azimuth3        ; fatal error...
        if      lineclr
        call    cline           ; clear line
        endif
        mvi     a,20            ; Display negative angle
        sta     column
        lxi     h,set$cur       ; set cursor
        call    text
        mvi     a,'-'           ; direction of angle
        call    co              ; output...
        mvi     a,' '           ; space
        call    co
        lhld    ypoint          ; translation
        mov     e,m             ; data negative...
        mvi     d,0
        call    pnum            ; display
        lxi     h,me5           ; print minutes
        call    text
        mvi     a,45            ; Display Positive angle
        sta     column
        lxi     h,set$cur
        call    text
        mvi     a,'+'           ; direction of angle
        call    co              ; output...
        mvi     a,' '           ; space
        call    co
        lhld    xpoint          ; translation
        mov     e,m             ; data positive..
        mvi     d,0
        call    pnum            ; display
        lxi     h,me5           ; print minutes
        call    text
;
azimuth2:
        call    cstat           ; abort?
        jz      azimuth1
        call    ci              ; test...
        jmp     azimuth1        ; no..
;
;       Fatal error Has occurred
;
azimuth3:
        call    fatal           ; fatal read error
        jmp     azimuth2
***********************************
*** "RADIAL ALIGNMENT CHECK" ***
***********************************
radial:
        lxi     h,rh$tbl        ; table pointer
        shld    temp
        lxi     h,me7           ; track display..
        mvi     b,'G'           ; limit
        call    inp$trk         ; select track and seek
        call    cls
        lxi     h,me8           ; print scale...
        call    text
        lxi     h,me11          ; Space bar msg
        call    text
```

(Continued on next page)

Diagnostic Diskette Listing

(Listing continued, text begins on page 40)

```
;
radial1:
;
            lxi     h,rad$pos       ; radial positive
            shld    xpoint          ; cursor table
;
            lxi     h,rad$neg       ; radial negative
            shld    ypoint          ; cursor table
;
            call    rd$trk          ; read radial track
            mvi     a,11            ; set row position
            sta     row
            jc      radial4         ; Fatal error has occurred
            call    cline           ; clear line (1)
            lda     row
            push    psw
            inr     a               ; clear line (2)
            sta     row
            call    cline
            pop     psw             ; reset line position
            sta     row
            lhld    ypoint          ; negative
            mov     a,m
            sta     column          ; position cursor
            mov     b,a
            lxi     h,set$cur
            call    text
            mvi     a,'^'           ; set pointer
            call    co
            lhld    xpoint
            mov     a,m             ; positive position
            sta     column
            mov     c,a             ; save cursor position
            lxi     h,set$cur       ; set cursor
            call    text
            mvi     a,'^'
            call    co
            mov     a,b
            sta     column          ; reset column position
            lda     row             ; down one line
            inr     a
            sta     row


***************************************
*** "SET POINTERS AND DRAW LINE" ***
***************************************

;           used to draw the graphic display of the
;           radial error.

            lxi     h,set$cur       ; position cursor
            call    text
            mov     a,c
            sub     b               ; line length
            dcr     a               ; less (1)
            mov     b,a
            mvi     a,'|'           ; pointer...
            call    co
;
radial2:
            mvi     a,'_'           ; jointing line
            call    co
            dcr     b               ; line length -1
            jnz     radial2
            mvi     a,'|'
            call    co
;
```

```
radial3:
            call    cstat           ; check for abort
            jz      radial1         ; read again
            call    ci              ; new track?
            cpi     ' '
            jnz     radial1         ; no...
            jmp     radial          ; new track...

****************************
*** "FATAL RADIAL ERROR" ***
****************************

radial4:

            if      lineclr
            call    cline           ; clear display
            lda     row             ; next line
            inr     a
            sta     row
            call    cline
            endif

            call    fatal           ; fatal read error
            jmp     radial3         ; abort?


*****************************
*** "CHECK SPINDLE SPEED" ***
*****************************
;
;           All timing is based on 4mhz Z80 Cpu
;
rpm:
            call    cls             ; clear display
            lxi     h,me19          ; test performed
            call    text
            lxi     h,me15          ; frame...
            call    text
            mvi     a,10            ; set row
            sta     row
;
rpm1:
            call    rpm2            ; compute time in 100us
                                      increments
            shld    temp            ; store count

            if      lineclr
            call    cline           ; clear line
            endif

            mvi     a,19            ; set cursor
            sta     column
            lxi     h,set$cur
            call    text
            lhld    temp
            xchg                    ; convert to milliseconds
            lxi     h,10
            call    div16
            push    h               ; Save remainder
            call    pnum
            mvi     a,'.'           ; decimal point
            call    co
            pop     h
            xchg
            call    pnum            ; fraction
            lxi     h,me20          ; milliseconds msg
            call    text
            mvi     a,50
            sta     column
            lxi     h,set$cur
            call    text
```

(Continued on next page)

Diagnostic Diskette Listing

(Listing continued, text begins on page 40)

```
        lhld    temp            ; convert to RPM
        mvi     a,9             ; divide By 600,000
        lxi     d,27c0h
        call    div24
        push    h               ; remainder
        call    pnum
        mvi     a,'.'           ; fraction
        call    co
        pop     h
        xchg
        call    pnum
        lxi     h,me21          ; rpm msg
        call    text
        call    cstat           ; abort?
        jz      rpm1            ; No..
        call    ci              ; Esc key?
        jmp     rpm1            ; No..

***********************************
*** "INDEX TO INDEX TIMING" ***
***********************************

;
;       With 4 Mhz. Z-80,
;       returns time in 100 Us increments to caller
;       in "HL" regs.

rpm2:
        call    clpend          ; Clear pending commands
        lxi     h,0             ; Clear Counter
rpm3:
        in      stat            ; Loop Until Index
        ani     2               ; Mask All But Index Bit
        jnz     rpm3            ; No Index
rpm4:
        in      stat            ; if Index...
        ani     2               ; Wait for No Index
        jz      rpm4            ; In Index Hole

;       Add index hole to count
rpm5:
        call    rpm7            ; 4.25
        in      stat            ; 2.75
        ani     2               ; 1.75
        jnz     rpm5            ; 2.50
                                ; Total = 11.25
;       Count until  next Index

rpm6:
        call    rpm7            ; 4.25
        in      stat            ; 2.75
        ani     2               ; 1.75
        jz      rpm6            ; 2.50
        ret                     ; total = 11.25

****************************
*** "RPM TIMING LOOP" ***
****************************
;
;       The loop time must be adjusted for 100us
;       increments at your clock speed with your processor.
;
;       note:   "RPM5 or RPM6" loop time must be counted.
;

rpm7:
        inx     h               ; 1.50
```

```
        mvi     a,22            ; 1.75
;
rpm8:
        dcr     a               ; 1.00  (22 * 1.00) = 22 Usec
        jnz     rpm8            ; 2.50  (22 * 2.50) = 55 Usec
        inx     d               ; 1.50
        inx     d               ; 1.50
        inx     d               ; 1.50
        inx     d               ; 1.50
        ret                     ; 2.50
                                ; total = 88.75



***********************
*** "READ DDD TRACK" ***
***********************
;
;       Att: Main read routine for program.
;            ==============================
;
;       Reads the Positive and Negative offset
;       sectors and returns pointers to the
;       translation tables set by caller.
;

rd$trk:
        MVI     A,7     ;Upper case code adds a simple display of
                        ;       last sector read (+ & -), giving
                        ;       ? and @ if first sectors fail
                        ;       and A thru Z representing the 26
                        ;       possible sectors on an 8" disk
                        ;       or A thru P for a 5.25" disk
        STA     ROW
        MVI     A,39
        STA     COLUMN
        LXI     H,SET$CUR
        CALL    TEXT
        mvi     c,lpos          ; last sector
        mvi     a,1             ; beginning sector positive
        call    rd$sec          ; read positive offsets
        ADI     3eH     ; translate sector # to alpha code */
        CALL    CO      ; display on console  */
        SUI     3eH     ; return to sector # */
        cpi     1               ; unable to read
        jz      rd$trk2         ; first sec?
        sta     diff            ; store last sector tested
        mvi     c,lneg
        mvi     a,2             ; beginning sector negative
        call    rd$sec          ; read negative offsets
        ADI     3eH
        CALL    CO
        SUI     3eH
        cpi     2               ; fatal error?
        jz      rd$trk2         ; set error flag
;
;       Positive sector Translation
;
        lda     diff
        inr     a               ; adj. sector Num.
        sui     4
        rar                     ; divide by two
        lhld    xpoint          ; translate Table pointer
        mvi     d,0             ; setup for Offset
        mov     e,a
        dad     d               ; add offset
        shld    xpoint          ; set pointer
;
;       Negative Sector Translation

        in      sec             ; last sector
        sui     4
```

(Continued on next page)

Diagnostic Diskette Listing (Listing continued, text begins on page 40)

```
        rar                     ; divide by two
        mov     c,a
        sub     e               ; diff. for Hysteresis
        sta     diff
        mov     e,c             ; offset
        lhld    ypoint          ; translation table pointer
        dad     d               ; offset in table
        shld    ypoint          ; set pointer
        ora     a               ; clear carry
        ret

;       Set fatal error flag, caller processes
;       the error.

rd$trk2:
        stc                     ; carry = Fatal error
        ret



*********************************************
*** "READ SECTORS BY INCREMENTS OF TWO" ***
*********************************************

;
;       Enter with first sector to be read in 'A' reg.
;       Returns to caller on Error or Last sector.
;       note:
;           Z flag set if good read else Nz
```

```
rd$sec:
        call    read            ; read sector
        in      sec             ; current sector
        rnz
        adi     2
        out     sec
        cmp     c               ; last sec?
        rz                      ; yes...
        jmp     rd$sec
********************************************************
***  "THE FOLLOWING ARE HARDWARE DEPENDENT ROUTINES"  ***
********************************************************
;
;
**********************
*** "DRIVE SELECT" ***
**********************
;
;       The drive select routine must be expanded
;       for selected systems.
select:
        if      Dysan
        mvi     a,21h           ; select bits
        out     dsel            ; select drive...
        endif
        if      CCS     ;The upper case code performs drive
                        ; selection on the CCS 2422 controller
                        ; for single side double density...
        CALL    CLS     ; clear console */
```

(Continued on next page)

Diagnostic Diskette Listing (Listing continued, text begins on page 40)

```
         LXI     H,SELMSG
         CALL    TEXT     ; print drive selection request */
         CALL    CI       ; get response */
         CPI     'A'      ; accept A,B,C,D, as responses */
         JZ      SELECT1
         JC      SELECT
         CPI     'B'
         JZ      SELECT1
         CPI     'C'
         JZ      SELECT3
         CPI     'D'
         JZ      SELECT4
         JMP     SELECT   ; repeat if invalid choice */


SELECT4:ADI      3        ; make low nybble an 8 */
SELECT3:INR      A        ; make a 4 */
SELECT1:SUI      '@'      ; make B into 2 or A into 1 */
         ORI     0F0H     ; add high nybble: */
                          ; F for double density, B for single
         STA     SELPT    ; store select bits for future use... */
         OUT     DSEL
         MVI     A,40H    ; select side 0...  00 selects side 1... */
         OUT     04       ; secondary control port */
         endif

         jmp     home              ; home drive

*****************************
*** "SEEK HEAD(S) TO TRACK" ***
*****************************
;
;        Enter with new track location in
;        'A' register.

track:
         out     data              ; new track
         call    clpend            ; clear FDC
         mvi     a,seek+rate       ; seek command + step rate
;
track1:
         out     cmd               ; issue...

         if      Dysan
         call    delay             ; delay for FDC
         endif

         if      CCS      ;This controller seems to need more
                          ; delay - I can't explain why... LA.
         call    delayB            ; delay for FDC
         endif

;
track2:
         in      stat              ; busy ?
         rrc                       ; test
         jc      track2            ; still busy...
         mvi     a,sdelay          ; set seek delay bit
         sta     dflag
         in      trk               ; set current track
         sta     ctrk
         ret

         if      CCS
DELAYB:  PUSH    B        ; extra delay for CCS controller... */
         LXI     B,2000H

DELAYC:  DCX     B
         MOV     A,C
```

```
        ORA     B
        JNZ     DELAYC
        POP     B
        RET
        endif


****************************************
*** "RESTORE HEAD(S) TO TRACK ZERO" ***
****************************************

home:
        call    clrend            ; clear FDC
        mvi     a,restore+rate    ; restore command + step rate
        jmp     track1            ; issue command...

******************************
*** "READ SECTOR ROUTINE" ***
******************************

;       Enter this routine with the sector to
;       be read in the 'A' register.
;
;       This routine will keep trying to read
;       the sector until retry count equals zero.
;
;       The 1793 controller has a built-in automatic
;       four retries to read ID, so the value set in
;       the equate (RETRY) is equal to [ 4 * retry + 1].

read:
        out     sec               ; set sector
```

(Continued on next page)

```
        mvi     e,retry         ; to read
;
read1:
        call    clpend          ; clear FDC
        call    prg$dma         ; program "DMA" controller
        if      CCS
        LHLD    DMA     ; get pointer to memory buffer */
        MVI     B,40H   ; initialize loop counter for I/O */
                        ;transfers: 40 for 8" DD, 20 for 8" SD
        endif
        lda     dflag           ; seek delay flag
        ori     rsec            ; read sector command
        out     cmd             ; issue..
        if      Dysan
        mvi     a,sdma          ; start transfer
        out     dma
        endif
        if      CCS
BREAD:  IN      DATA    ; code to perform I/O transfer of  */
        MOV     M,A     ;       data to memory */
        INX     H
        IN      DATA
        MOV     M,A
        INX     H
        IN      DATA
        MOV     M,A
        INX     H
        IN      DATA
        MOV     M,A
        INX     H
        DCR     B       ; decrement loop counter... */
        JNZ     BREAD
        endif
        call    comp            ; transfer completed?
;
read2:
        xra     a               ; clear seek delay
        sta     dflag
        in      stat            ; good read?
        ora     a
        rz                      ; yes...
        dcr     e               ; count off retries
        jnz     read1           ; try again
        in      stat            ; return status
        ora     a
        ret
****************************************
*** "WAIT FOR END OF DMA TRANSFER" ***
****************************************
comp:
        call    delay           ; before each check
        in      stat            ; Completed Transfer?
        rrc
        rnc                     ; YES!
        jmp     comp
**********************************
*** "PROGRAM DISK CONTROLLER" ***
**********************************
```

```
prg$dma:

        if      Dysan
        mvi     b,15            ; Bytes Count
        lxi     h,cmd$tbl       ; Command Table
;
prg$dma1:
        mov     a,m             ; load
        out     dma             ; write to controller
        inx     h
        dcr     b               ; -1
        jnz     prg$dma1
        endif

        if      CCS
        LXI     H,SECBUF        ; store address of memory buffer */
        SHLD    DMA     ; in pointer for disk controller */
        LDA     SELPT   ; get drive select bits */
        OUT     DSEL    ; sets autowait for 2422 controller... */
                        ; to ensure synchronization of I/O transfer
        endif

        ret

****************************************
*** "COMMAND TABLE DMA CONTROLLER" ***
****************************************

;       This data is used to program Zilog's
;       z80 DMA controller.
```

(Continued on next page)

Diagnostic Diskette Listing (Listing continued, text begins on page 40)

```
cmd$tbl:

        if      Dysan
        db      0c3h            ; Re-Set DMA Controller
        db      8bh             ; Clear Block Counter
        db      79h             ; Recieve Block
        dw      secbuf          ; Address To Store Block
        dw      256             ; Block Size
        db      14h             ; Define Port (B) Address
        db      28h             ; Define Port (A) Address
        db      85h
        db      data            ; Data Reg FDC
        db      8ah             ; Set DMA Controller Active HIGH
        db      0cfh
        db      3               ; Data  ----> Memory
        db      0cfh
        endif


*************************************
*** "CLEAR PENDING COMMANDS FDC" ***
*************************************
;
;       Clears the floppy controller of any
;       pending commands.

clpend:
        mvi     a,clear         ; clear FDC command
        out     cmd             ; issue...

****************************************
*** "FDC DELAY TO PROCESS COMMANDS" ***
****************************************
;
;       This delay must be adjusted for different
;       clock speeds. 50 usec delay loop.
;
;       Note:
;            If altered, Ajust Index Timing Routine.

delay:                          ;  4.25
        mvi     a,fdelay        ;  1.75
delay1:
        dcr     a               ; 11.00
        jnz     delay1          ; 27.50
        nop                     ;  1.00
        nop                     ;  1.00
        nop                     ;  1.00
        ret                     ;  2.50

********************************
*** "RADIAL TRANSLATE TABLE" ***
********************************

;       Cursor Positioning table for radial
;       alignment test.

rad$neg:
        db      38              ; 1  millinch
        db      36              ; 2  "    "
        db      34              ; 3  "    "
        db      32              ; 4  "    "
        db      30              ; 5  "    "
        db      28              ; 6  "    "
        db      26              ; 7  "    "
        db      24              ; 8  "    "
        db      22              ; 9  "    "
```

(Continued on page 76)

GGM-FORTH™ has HELP*
for Z80† using CP/M†

GGM-FORTH, a software system for real-time measurement and control, runs on any Z80 computer under CP/M, using an extended fig-FORTH vocabulary.

GGM-FORTH includes:

- Up to four open CP/M files, in any combination of direct-access and sequential-access, fully compatible with all CP/M utilities
- Char. in/out uses CP/M console, lister, file, or auxiliary port

PLUS

- On-Line HELP* facility, compliments our complete manual
- Separate HELP* files for multiple, user-extendable vocabularies
- Full - Screen editor uses HELP* for better programmer productivity

Complete system and manual: $150.

GGM SYSTEMS, INC. (617) 662-0550
135 Summer Ave. Reading, MA 01867

†Z80 is a trademark of Zilog, Inc.
†CP/M is a trademark of Digital Research, Inc.

Circle **no. 42** on reader service card.

PROMPT DELIVERY! AND TRY TO BEAT THESE IC PRICES:

DYNAMIC RAM		
64K	200 ns	$ 5.65
64K	150 ns	5.95
64K	120 ns	6.27
16K	200 ns	1.56
EPROM		
27128	300 ns	$19.20
2764	250 ns	5.99
2732	450 ns	4.25
2716	450 ns	3.60
2532	450 ns	4.75
STATIC RAM		
6264P-15	150 ns	$38.00
6116P-3	150 ns	5.62
6117P-3	150 ns	4.20

Prices 10/18/83

MasterCard/VISA or UPS CASH COD
Factory New, Prime Parts
μP∞
MICROPROCESSORS UNLIMITED
24,000 South Peoria Ave.
BEGGS, OK. 74421
(918) 267-4961

Prices subject to change. Please expect higher prices on some parts due to world wide shortages. Call for volume prices. Subject to available quantities. Shipping & Insurance extra. Cash discount prices shown. Federal Express Standard Air @ $5.99! Orders received by 5 PM CST can be delivered to you by the next morning.

Circle **no. 67** on reader service card.

NEW IBM-PC DEBUGGER

And now... right before your very eyes...

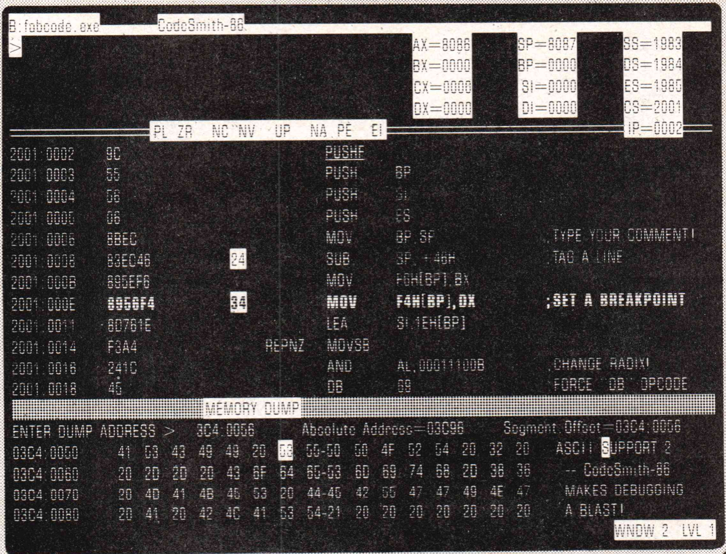
CodeSmith™-86

New full screen debugging concept. With unique multi-window multi-level split-screen features:

- Simple/single keystroke commands
- Full-screen disassemblies may be scrolled through
- Continuous monitoring of selected memory
- Hundreds of tagged breakpoints supported
- Saves and restores user's graphic display when breakpoint hit
- Many other inspired features

Your bugs were never so obvious!

Requires MS-DOS & 96K RAM (min.).



Designed to get your program working—FAST.

INTRODUCTORY OFFER ONLY
California residents add 6.5% sales tax
$145.00

VISUAL AGE
642 N. Larchmont Blvd., Los Angeles, CA 90004 • (213) 464-8141

CodeSmith is a registered trademark of International Arrangements, Inc.
Microsoft and MS are registered trademarks of Microsoft Corp. • IBM is a registered trademark of International Business Machines Corp.

Circle **no. 120** on reader service card.

Diagnostic Diskette Listing (Listing continued, text begins on page 40)

```
                db      20                      ; 10 "     "
                db      17                      ; 11 "     "
                db      14                      ; 12 "     "
                db      11                      ; 13 "     "

rad$pos:
                db      40                      ; 1  millinch
                db      42                      ; 2  "       "
                db      44                      ; 3  "       "
                db      46                      ; 4  "       "
                db      48                      ; 5  "       "
                db      50                      ; 6  "       "
                db      52                      ; 7  "       "
                db      54                      ; 8  "       "
                db      56                      ; 9  "       "
                db      58                      ; 10 "       "
                db      61                      ; 11 "       "
                db      64                      ; 12 "       "
                db      67                      ; 13 "       "

*********************************
*** "AZIMUTH TRANSLATE TABLE" ***
*********************************

azi$tab:
                db      18                      ; head azmuith angle
                db      20                      ; +/- minutes
                db      22
                db      24
                db      26
                db      28
                db      30
                db      32
                db      34
                db      36
                db      38
                db      40
                db      42

**************************
*** "PROGRAM MESSAGES" ***
**************************

me1:

                db      (-1) AND 0FFH,18,5,'<*>==================================
                db      (-1) AND 0FFH,18,6,'<*>         DIAGNOSTIC COMMAND MENU
                db      (-1) AND 0FFH,18,7,'<*>==================================
                db      (-1) AND 0FFH,18,8,'<*>
                db      (-1) AND 0FFH,18,9,'<*>   R = Radial        H = Hysteresis
                db      (-1) AND 0FFH,18,10,'<*>
                db      (-1) AND 0FFH,18,11,'<*>   A = Azimuth       I = Index Timi
                db      (-1) AND 0FFH,18,12,'<*>
                db      (-1) AND 0FFH,18,13,'<*>   C = Centering     S = Spindle Sp
                db      (-1) AND 0FFH,18,14,'<*>
                db      (-1) AND 0FFH,18,15,'<*>   D = Drive Sel     E = Exit Progr
                db      (-1) AND 0FFH,18,16,'<*>
                db      (-1) AND 0FFH,18,17,'<*><*><*<*><*><*><*><*><*><*><*><*><
                db      (-1) AND 0FFH,18,19,'SELECTION? ....',bs,0

me2:            db      'FATAL READ ERROR         ',0
me3:            db      'RE-CLAMP DISKETTE ....',bs,0
me4:            db      'CENTERING OK ....        ',bs,0
me5:            db      '  Minutes                ',0

me7:            db      (-1) AND 0FFH,25,2,'<>==========================<>'
                db      (-1) AND 0FFH,25,3,'<>      TRACK SELECTION     <>'
                db      (-1) AND 0FFH,25,4,'<>==========================<>'
```

(Continued on page 78)

```
              db      (-1) AND 0FFH,25,5,'<>                              <>'
              db      (-1) AND 0FFH,25,6,'<>    A = (0)       D = (41)   <>'
              db      (-1) AND 0FFH,25,7,'<>                              <>'
              db     (-1) AND 0FFH,25, 8,'<>    B = (3)       E = (70)   <>'
              db     (-1) AND 0FFH,25, 9,'<>                              <>'
              db     (-1) AND 0FFH,25,10,'<>    C = (38)      F = (73)   <>'
              db     (-1) AND 0FFH,25,11,'<>                              <>'
              db     (-1) AND 0FFH,25,12,'<>==========================<>'
              db     (-1) AND 0FFH,25,13,'<>     <ESC> Cancels Test     <>'
              db     (-1) AND 0FFH,25,14,'<>==========================<>'
              db     (-1) AND 0FFH,25,16,'TRACK? ...',bs,0
me8:          db      (-1) AND 0FFH,24,5,'---  RADIAL ALIGNMENT CHECK ---'
              db      (-1) AND 0FFH,10,9,'Away',(-1) AND 0FFH,36,9,'Spindle',(
              db      (-1) AND 0FFH,10,10,'13 12 11 10 9 8 7 6 5 4 3 2 1'
              db              '-1 2 3 4 5 6 7 8 9 10 11 12 13',0
me9:          db      (-1) AND 0FFH,25,2,'<>==========================<>'
              db      (-1) AND 0FFH,25,3,'<>       TRACK SELECTION      <>'
              db      (-1) AND 0FFH,25,4,'<>==========================<>'
              db      (-1) AND 0FFH,25,5,'<>                            <>'
              db      (-1) AND 0FFH,25,6,'<>    A = (35)    B = (44)    <>'
              db      (-1) AND 0FFH,25,7,'<>                            <>'
              db      (-1) AND 0FFH,25,8,'<>          C = (47)          <>'
              db      (-1) AND 0FFH,25,9,'<>                            <>'
              db     (-1) AND 0FFH,25,10,'<>==========================<>'
              db     (-1) AND 0FFH,25,11,'<>    <ESC>  Cancels Test     <>'
              db     (-1) AND 0FFH,25,12,'<>==========================<>'
              db     (-1) AND 0FFH,25,14,'TRACK? ...',bs,0
me10:         db      (-1) AND 0FFH,23,5,'---  DISKETTE CENTERING CHECK ---',0
me11:         db      (-1) AND 0FFH,23,14,'Press <SPACE BAR> For New Track',0
me12:         db      (-1) AND 0FFH,24,5,'--- AZIMUTH ALIGNMENT CHECK ---'
              db      (-1) AND 0FFH,25,14,'<ESC> Will Cancel AZIMUTH Check',0
me13:         db      (-1) AND 0FFH,25,2,'<>==========================<>'
              db      (-1) AND 0FFH,25,3,'<>       TRACK SELECTION      <>'
              db      (-1) AND 0FFH,25,4,'<>==========================<>'
              db      (-1) AND 0FFH,25,5,'<>                            <>'
              db      (-1) AND 0FFH,25,6,'<>    A = (0)     B = (76)    <>'
              db      (-1) AND 0FFH,25,7,'<>                            <>'
              db      (-1) AND 0FFH,25,8,'<>==========================<>'
              db      (-1) AND 0FFH,25,9,'<>     <ESC> Cancels Test     <>'
              db     (-1) AND 0FFH,25,10,'<>==========================<>'
              db     (-1) AND 0FFH,25,12,'TRACK? ...',bs,0
me15:         db      (-1) AND 0FFH,15,8,'######################################
              db     (-1) AND 0FFH,15,12,'######################################
me19:         db      (-1) AND 0FFH,27,5,'--- SPINDLE SPEED CHECK ---'
              db      (-1) AND 0FFH,25,14,'<ESC>  Will Cancel RPM Check',0
me20:         db      '  Milliseconds',0
me21:         db      '  RPM',0
BLANKS:       DB      '
              DB      '                    ',0
SELMSG:       DB      (-1) AND 0FFH,15,12,'WHICH DRIVE HAS THE DDD - A,B,C,D ?
              DB      (-1) AND 0FFH,25,14,'SELECTION ? ....',0
*************************
*** "POSITION CURSOR" ***
*************************
set$cur db         0ffh
```

```
column  ds      1                       ; set column
row     ds      1                       ; set row
        db      0                       ; terminator
**********************************
*** "PROGRAM VARIABLE STORAGE" ***
**********************************
temp    ds      2                       ; temp storage
diff    ds      1                       ;  1st - 2nd read
ctrk    ds      1                       ; current track
savtrk  ds      1                       ; temp storage Hysteresis test
dflag   ds      1                       ; seek delay flag
point   ds      2                       ; table pointer
xpoint  ds      2                       ; table pointer positive
ypoint  ds      2                       ; table pointer negative
xoff    ds      1                       ; positive offset
yoff    ds      1                       ; negative offset
bais    ds      1                       ; bais from test track (Hysteresis)
hyerr   ds      1                       ; hysteresis error on first reading
SELPT   DS      1         ;STORES BYTE FOR DISC CONTROL PORT
;
;       Storage area for 16/24 bit divide routine
;
msb     ds      1
msb1    ds      2
msb2    ds      2
inbuf:  db      3                       ;"max" byte for BDOS 10
incnt   ds      1                       ;"cnt" buffer for BDOS 10
buffer  ds      4                       ; console input buffer
secbuf  ds      256                     ; sector buffer
        ds      100                     ; stack space
stack   equ     $                       ; top down...
```

End Listing

Building a Programmable Frequency Synthesizer

The Master Controller Board (MCB), a single-board computer available from Space-Time Productions, Chicago, Illinois, has an 8253 counter/timer on it that is used for a baud rate generator. Since it has two spare counter time channels, I decided to use them to construct a crystal-controlled programmable frequency synthesizer.

Phase-Locked Loop (PLL) Theory

A phase-locked loop is a very useful combination analog and digital device. It enables a digital frequency divider to multiply frequencies exactly. It performs this trick by controlling the frequency and phase of a voltage-controlled oscillator (VCO) in such a wav that the VCO is locked into the frequency and phase of a reference oscillator. (See Figure 1, below.)

by Michael L. Simon

Michael L. Simon, Space-Time Productions, 2053 N. Sheffield, Chicago, Illinois 60614.

An example will help to explain the operation. A reference frequency of 1000 Hz is available; it is used as one input to a phase comparator. The other input comes from the output of the VCO through a programmable divider, which is set to divide by four. For the loop to be locked, the output of the programmable divider must be 1000 Hz. If the frequency is low, the phase comparator will generate longer pulses which, when averaged by the low pass filter, will create a higher voltage at the input of the VCO. This will cause the output of the VCO to increase in frequency, maintaining an output of 4000 Hz.

Similarly, if the output of the VCO is above 4000 Hz, the phase comparator will generate shorter pulses. This will reduce the frequency, again restoring the original 4000 Hz. Thus a phase-locked loop and a divide-by-four circuit are able to multiply an input frequency by four. If the input to the reference of the loop has high stability, the output of the loop will also tend to have high stability.

Figure 2 (page 81) shows the circuit diagram of a phase-locked loop.

The 4046 CMOS Phase-Locked Loop Chip

The 4046 is a CMOS phase-locked loop which is fairly inexpensive and operates over a wide frequency range (from less than 1 Hz to over 1 million Hz). The 4046 has a phase detector that is edge sensitive. This is useful when the signals to be synchronized are not perfect square waves. The chip has a second phase detector that can be used to tell if the loop is out of lock. Because the 4046 is a CMOS device with high input impedance, loop filters with high value resistors and low value capacitors are possible. This makes possible low-frequency loop filters built from small, inexpensive capacitors.

The Single-Board Computer

The MCB is a Z80-based, single-board computer with many useful features. Among these features are 72 parallel I/O lines, two serial I/O ports, and two counter/timer chips with a total of eight counter/timers. Also available for the board is a version of the TDL monitor, modified to allow control of the programmable dividers of the 8253. This program makes frequency synthesizer experiments a snap since it has commands that directly control the dividers in the 8253 counter/timer chip.

One channel of the 8253 is used for the on-board baud rate generator. Each channel of the baud rate generator is set up for a different baud rate; three rates

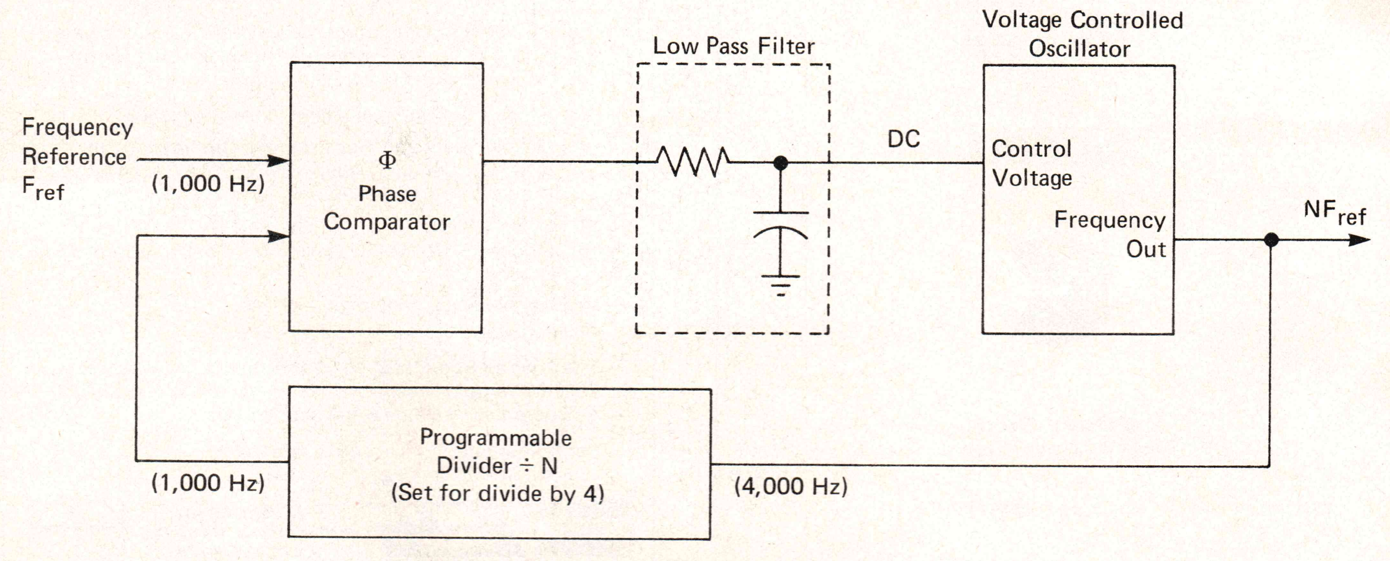


Figure 1.

Phase-locked loop (PLL) example.

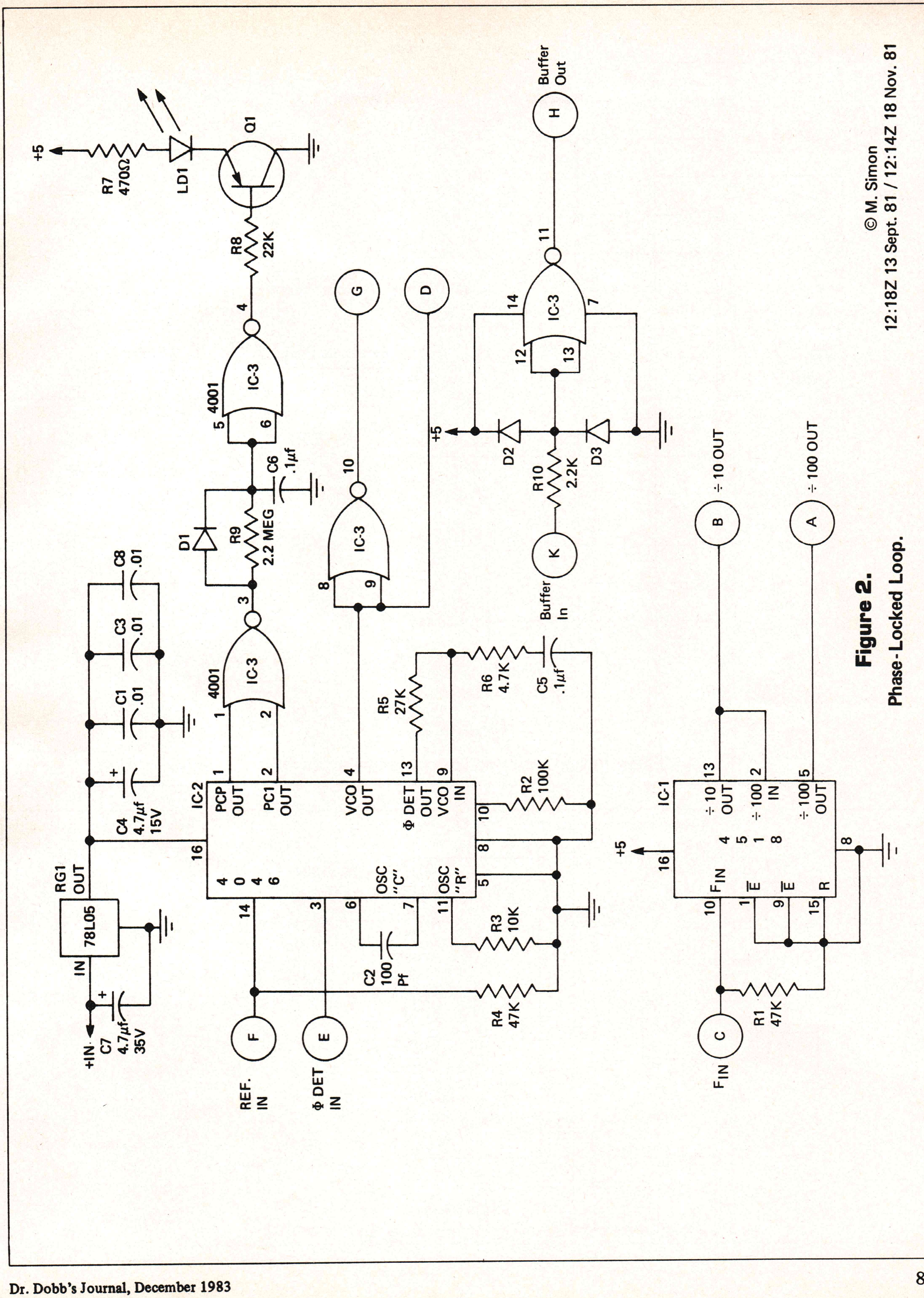


Figure 2.
Phase-Locked Loop.

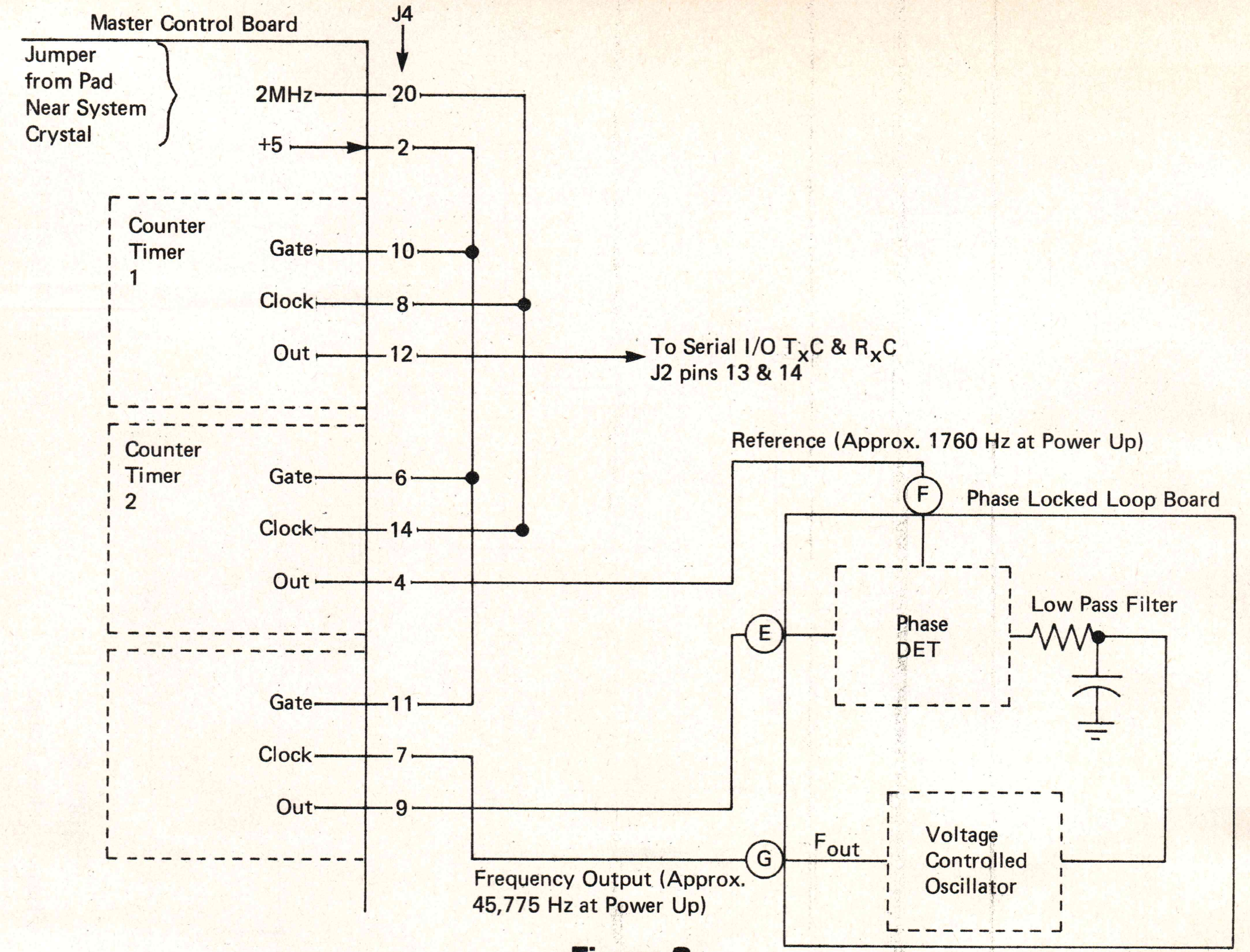


Figure 3.

Phase-locked loop using a programmable divider.

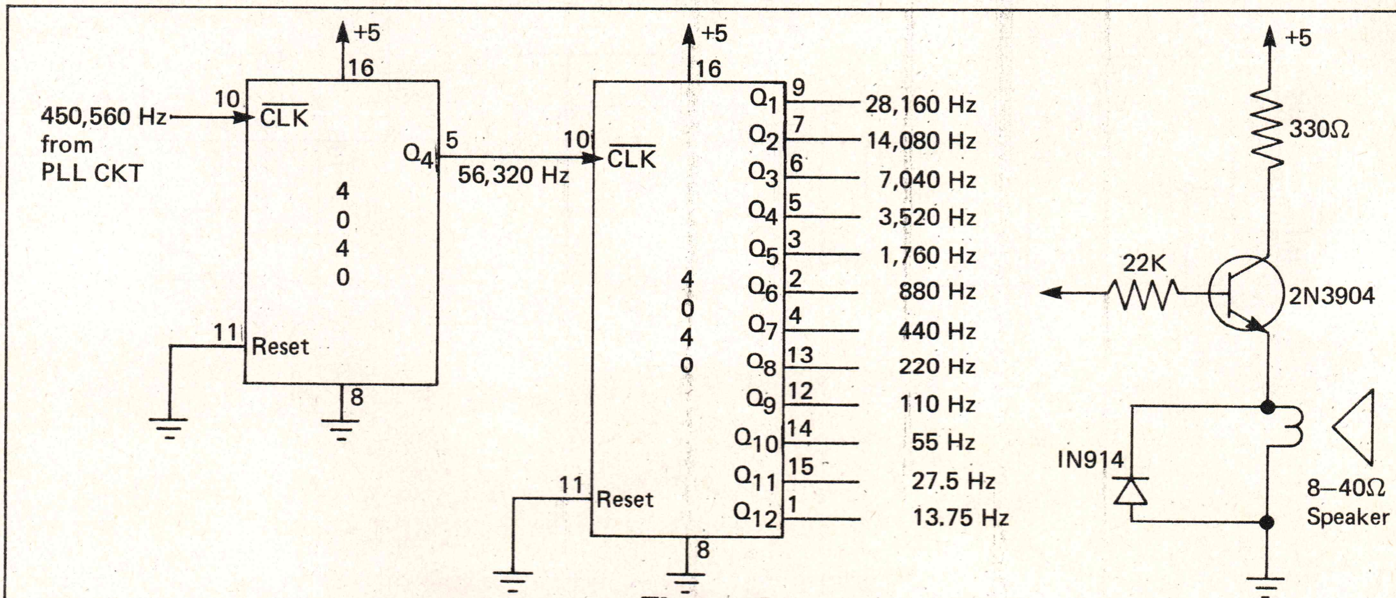


Figure 4.
Programmable PLL-controlled musical instrument tuner.

(110, 300, and 4800) are available. Since I use 300 baud to talk to the MCB, which uses channel 1 of the 8253, channels 0 and 2 are available for experiments. The 8253 is set up by the monitor program to act as a programmable divide-by-N circuit and to produce a square wave. If N is an even number, the output will be a square wave. If N is an odd number, the output will be approximately a square wave with one half of the cycle longer than the other half; refer to Figure 3 (page 82) and Listing One (page 88).

The gate of each channel used must be pulled high for the 8253 to perform the divide function. A 2 MHz buffered, crystal-controlled reference signal is available from the CPU's clock generator. This signal is required by the baud rate generator and is brought to pin 20 of header J-4 from the pads near the crystal by a jumper wire soldered to the underside of the board. From this pin I used reuseable jumper wires on the header pins to bring the 2 MHz signal to the inputs of counters 0, 1, and 2. These jumper wires are about the handiest I have ever used on .025 square pins. They make wrapped wire methods seem very clumsy, especially when doing experiments, since the socketed jumpers are easy to move and reuse.

After hooking up the PLL according to diagram 2, I expected an output frequency of about 45,775 Hz – I say "about" because the output frequency depends on the exact frequency of the CPU time base generator and the divide ratios chosen for the baud rate generators. My oscillator was slightly low, so I got a different value.

I prepared a chart of divide ratios versus hexadecimal numbers. The baud rate generators could be programmed to recognize decimal values. However, by using hexadecimal numbers, over six times the range of divisors is possible.

My first experiment was to get the system to go out of lock on the low end. The frequency at which this happens depends on the components used in the loop filter. With the components chosen, the loop goes out of lock at a reference frequency of about 200 Hz into the phase comparator. The exact frequency will vary with the rate at which the PLL has to change between new inputs. The loop goes out of lock on a jump from divide-by-1000 hex to divide-by-2000 hex on the channel 2 counter. However, if you can inch up to 1F00 and keep the loop in lock, it is possible to jump from there to 2000 hex and still keep the loop in lock.

The same method is used for getting the loop back in lock from an out-of-lock condition. To get the loop to respond to lower frequencies and still stay in lock, try increasing R5 and C5. (Check out *Design of Phase Locked Loop Circuits With Experiments* by Howard M. Berlin, Howard W. Sams Inc., for methods of determining the best values for the loop filter.)

On the high end it is possible to keep the loop in lock until well past 1 MHz. However, I have had some PLL's lock up at the high frequencies. This is probably due to the speed limitation of the chips I used. By removing power and beginning again, everything was returned to normal.

Solid State Scientific of Montgomery, Alabama, makes a version of the 4046 with a VCO guaranteed to oscillate up to 5 MHz. The part is called a 4446. By reducing capacitor C2 to 50 pf, I got mine to stay in lock past 2.5 MHz. For those who want to go past 2.5 MHz, I recommend using about 20 pf at C2. You must also use a divider capable of higher frequency input than the 8253. Fortunately, the Master Controller Board has a socket for the AMD 9513 counter/timer, which is guaranteed up to 7 MHz.

It is possible to generate almost any frequency with the right choice of dividers. For example, I wanted to tune up my AM radio. To do this I needed to tune up the

intermediate stages first. This requires a frequency of 455,000 Hz. With the crystal in my board, loading counter 0 with 3A4 hex and counter 2 with 1000 hex gave me a readout on my frequency counter of 454,995.0 Hz. Not bad, but I wanted to get closer.

The first step was to determine the frequency of the clock on the MCB. Checked with my trusty calculator, 3A4 converted to decimal is equal to 932; 1000H is 4096. Dividing 454,995 Hz by 932 found the reference frequency. Multiplied by 4096, the result was the crystal-controlled input clock frequency. My result was 1,999,634.6 Hz.

The dilemma now was to determine which divider ratios would yield the correct output frequency. There is a formula for doing this. Unfortunately, I had long since forgotten it, so I invented a trial-and-error method to determine the correct divisors. Since computers are much faster at trial and error than humans, I wrote a BASIC program (see Listing Two, page 91) to handle it. If I had known the right algorithm, there would have been quicker results, but the program does the job.

The main idea is that the phase-locked loop may be used to generate any desired frequency to any degree of accuracy by choosing the right values for the controlling dividers. By using this program, I came up with values of 808 for divider #0 and 3551 for divider #2. This gives an accuracy of 0.13 ppm. I converted 808 and 3551 to hexadecimal (328h and 0DDFh) and entered the values in the 8253. 455,000.0 Hz was the result!

With suitable dividers following the loop output, audio tones can be generated. These could be used to fine tune your favorite string instrument (see Figure 4, page 82). The 4040 circuit gives output ranging from the lowest to the highest octave, evenly spaced at octave intervals. The even-tempered scale used in most musical instruments uses twelve notes per octave interval. Each note is higher in frequency than the preceding note by a factor of 1.0594631. Table I (below) gives a series of frequencies that facilitates tuning all the notes in any octave. The beauty of this set-up is that with the proper choice of dividers, any scale may be reproduced, regardless of the number of notes or the size of the interval.

The Phase-Locked Loop board is available as a kit for $28.75 postpaid from Space-Time Productions, 2053 N. Sheffield, Chicago, Illinois 60614; (312) 327-0391. The Master Controller Board is available bare for $54.90 with postage. The customized TDL Monitor is available for $69.90. A partially populated MCB, assembled and tested, running Tiny Controller Basic, is $304.90 with postage.

DDJ

(Listing begins on page 88)

Note	Ratio	Input to 4040 Divider	A=440
A	1.0000000	450560.00 Hz	440.00000 Hz
A#	1.0594631	477351.69 Hz	466.16375 Hz
B	1.1224620	505736.47 Hz	493.88327 Hz
C	1.1892071	535809.15 Hz	523.25112 Hz
C#	1.2599210	567670.00 Hz	554.36523 Hz
D	1.3348398	601425.42 Hz	587.32951 Hz
D#	1.4142135	637188.03 Hz	622.25393 Hz
E	1.4983071	675077.20 Hz	659.25507 Hz
F	1.5874010	715219.44 Hz	698.45648 Hz
F#	1.6817928	757748.56 Hz	739.98882 Hz
G	1.7817974	802806.63 Hz	783.99084 Hz
G#	1.8877486	850544.00 Hz	830.60937 Hz
A	2.0000000	901120.00 Hz	800.00000 Hz

Table 1
Twelve Tone Scale Frequencies

EP-PLOT©

EP-PLOT: The Plot Program that is also a Plotter

EP-PLOT converts your MX-80 or FX-80 with GRAPTRAX into a high resolution digital plotter with the same command language as a Houston Inst. DM/PL+ plotter plus much more. Produce high resolution plots without an expensive plotter. If you have CP/M-80, an MX-80 or FX-80, with EP-PLOT, you can now have full plotting capability for $180.00.

EP-PLOT $180.00
Demonstration Disk $15.00
(refundable with system purchase)
Available for CP/M in 8' SSSD and 5' NorthStar SSDD

Order Now — Send Check or Money Order to:

SYMBOLIC SYSTEMS©
3138 Via Loma Vista, Escondido, CA 92025
Mail to: P.O. Box 1825, Escondido, CA 92025
(Calif. residents add 6% sales tax)

GRAFTRAX is a reg. trademark of EPSON-AMERICA Inc.
DM/PL+ is a reg. trademark of Houston Instrument Inc.
CP/M is a reg. trademark of Digital Research Inc.

Circle no. **100** on reader service card.

TALISMAN T.M.

It's almost magic!

CP/M® Terminal Translation
& Multiple Keyboard Redefinition Program

- ★ Emulates ANY interactive terminal — runs ANY CP/M 2.2 software on your microcomputer.
- ★ Bridges between your micro and mini or mainframe terminals.
- ★ Redefines any key(s) to produce up to 249 characters.
- ★ Creates, saves, edits, and retrieves up to 255 keyboard overlays.
- ★ Reprograms *on-the-fly*, while you're running another program.
- ★ And so much more!
- ★ A godsend for programming, data entry, word-processing.
- ★ Transparent to the user. No wiring or soldering. User-friendly. Best documentation in the industry.

TALISMAN™ is only $125, manual included.
California residents add 6% sales tax.

Call or write for free details.

disco tech®

DISCO-TECH®
a division of Morton Technologies, Inc.
600 B St. / P.O. Box 1659
Santa Rosa, CA 95402
Tel. 707 / 523-1600
Dealer inquiries invited.

Circle no. **34** on reader service card.

SOURCE CODE INCLUDED

B + TREE*

The **SOFTWARE 99**™ B + TREE indexing system is a BASICA Source Code implementation of file maintenance procedures for record modification, insertion, deletion and query. Featured is the description of integration procedures necessary to couple the B + TREE with application programs.

CROSS REFERENCE/ DOCUMENTATION UTILITY*

The need for a structured program listing & cross reference is satisfied using the **SOFTWARE 99**™ CROSS REFERENCE/ DOCUMENTATION UTILITY. Line numbers and variable names are cross referenced against their usage within a BASICA program. The listing is in an easily read form with one statement per line and structuring of IF statements and LOOPS. Output can be selected to the screen, printer or ASCII file.

GENERAL LEDGER*

The **SOFTWARE 99**™ GENERAL LEDGER is a remarkably easy to use, but fully integrated system of 9 BASICA programs run from a main menu. Some of these programs include Account Maintenance, Transaction Entry, Trial Balance, Balance Sheet and Income Statement. For those interested in Source Code, the 9 programs are all written in the same style, so that a study and understanding of one program gives you an understanding of each program.

*MINIMUM SYSTEM REQUIREMENTS:
64K, PC-DOS or MS-DOS
(any version), 1 disk drive

ONLY $99 EACH

Please add $6 for postage and handling.
VISA Calif. residents add 6% tax MasterCard

SOFTWARE 99™
THE SOURCE FOR SOURCE CODE

702 Washington Street #174
Marina del Rey, CA 90291
PHONE ORDERS **(213) 399-7463** 24 HOURS

PC-DOS a trademark of IBM Corporation
MS-DOS a trademark off Microsoft Corporation

Circle no. **95** on reader service card.

MEGA-BYTE

SPECIALIZING IN HARD DISKS AND HIGH PERFORMANCE STREAMING TAPE UNITS

MD-10

11 MEGABYTE (FORMATTED) 5 1/4" HARD DISK Including POWER! $1995

POWER!

Check out InfoWorld's Rave review of POWER Nov. 8/82)

55 CP/M UTILITIES IN ONE PACKAGE

...The super program that puts you in control of CP/M (or MS-DOS) and your winchester.

POWER automatically numbers disk files. Just pick file number to Copy, Erase, Reclaim, Type, etc.
...Your computer feeds the file names automatically. You do NO typing! NO typing error ever!
YOU DON'T NEED SYSTEM DISK IN ANY DRIVE.
No more BDOS ERROR!

YOU Test and Fix bad disks! Reclaim accidently erased files or programs! Single step thru memory up or down! Search, view, change memory or disk in a snap! See Status and File Size instantly! Verify check-sums for programs! Load or Save programs at any address.

55 prompted user-friendly funstions for housekeeping and a 120 page easy-read user's guide make POWER your most often used software. You'll use it every day!

Now, POWER! also includes a special program that lets you lock sensitive files, so that only you can access then. Without the secret PASSWORD which you can create and change at will, no prying eyes will ever know your secret file even exists.

SOME MAJOR POWER COMMANDS:

REN	CM	WRITEGR	DISK	STAT
TYPEA	USR 2	DUMPA	SPEED	SETDIR
SIZE	ERA	SEARCH	WRITE	RECLAIM
XUSER	TYPEX	?	DUMPX	DS
SETRO	CHECK	COPY	FILL	READ
GROUP	TEST	TYPEH	EX	DUMP
SAVE	SETWR	EXIT	DIR	MOVE
READGR	LOG	USER	TYPE	JP
DUMPH	LOAD	SETSYS	RUN	

For this low price your winchester will be delivered completely assembled and tested, with drive controller, case, power supply, cabling, Z-80 interface and the best disk controller software on the market.

- The unique and simple interfacing system does not tie up existing ports.
- Significantly faster than other winchester subsystems which interface through the IEEE-488 port.
- CP/M drivers require minimum memory overhead (about 2K-other systems require as much as 6k)
- The MD-10 can read or write a 64K file in less than four (4) seconds.
- A network system will be available shortly which can support up to sixteen mixed types of computers from one MD-10 or larger disk subsystem.
- With POWER! Software files can be code word protected.
- Other systems available: 22 and 44 megabytes

MD-20 with 22MB formatted: $2895

MD-44 with 44MB formatted: $3995

- Up to eight (8) winchester subsystems can be interfaced to one computer.
- Software supports 32 different user areas per MD-10.
- Backing up hard disk files is simple with the special software which is provided with all subsystems.
- With a single hard disk installation, the MD-10 subsystems becomes units A and B with the standard drives being designed E and F if a second MD-10 is installed later it becomes units C and D.
- MD-10 or larger systems will interface with IBM PC or any Z-80 computer (CCS, APPLE (CP/M), ZENITH/HEATH, NORTHSTAR, GODBOUT, XEROX 820, Z-80/S100, ALSPA, or TRS-80 MOD II) using CP/M, OASIS, PCDOS 2.0, and Godbout Software supports both CP/M and MP/M816.
- Double density modifications are available allowing you to later increase the capability of an MD-10 to about 20 megabytes, MD-20 to 40, etc.
- **Full One-Year Warranty**

HIGH PERFORMANCE STREAMING TAPE

Unique! Simple! Compatible!

- **Field-proven Archive Sidewinder tape unit.**
- **Interfaces with ANY SASI/SCSI compatible hard disk peripheral.**
- **Full CRC error-checking**
- **Superb reliablity.**
- **Full One-Year Warranty.**
- **Fast (up to 20MB in less than 5 minutes, up to 40MB in less than 10 minutes).**

$1795.00 20MB

$1995.00 45/60MB

$29.95 3M DC300XL Streaming Tape Cartridges

MEGA-BYTE

ORDER

Name	Qty.	Price	Total

VISA ☐ M/C ☐ COD ☐ Total Order ______

Card # ______ Exp. Date ______ If Utah Tax ______

Signature ______ Total Enclosed ______

TO ORDER OR FOR MORE INFORMATION:
(801) 257-7033 or mail to MEGA-BYTE INC. RT. 3 Box 35, Tremonton, Utah 84337
TERMS: Cashiers Check, VISA, M/C or COD.
Shipping charges added to all orders.

NAME ______
COMPANY ______ TITLE ______
ADDRESS ______ CITY ______
STATE ______ ZIP ______
TEL ( ) ______

CP/M is a registered trademark of Digital Research. OASIS is a proprietary product of Phase One Systems Inc. Z-80 is a trademark of Zilog. MD-10 is a registered trademark of Media Distributing.

Circle no. 64 on reader service card

Frequency Synthesizer

(Text begins on page 80)

Listing One

```
; 11:58z 22 MARCH 83 CT FOR PLL ART

CDL's Z80 Macro Assembler C12012-0312

Modified for MASTER CONTROLLER BOARD 21 MARCH 83  M.Simon

                        ; EQUATES FOR THE 8253 TIMER CHANNELS

  006C                   CTR0    == 6Ch                 ;8253 CHAN 0
  006D                   CTR1    == CTR0+1
  006E                   CTR2    == CTR0+2
  006F                   CTRTC   == CTR0+3              ;8253 CONTROL
                                                        ; CHANNEL


                         ;THE "BAUD" PROGRAM GETS TWO PARAMETERS OFF THE
                         ; STACK. THE FIRST OFF THE STACK IS A TWO BYTE
                         ; HEX NUMBER THAT IS THE DIVIDER THAT IS TO BE
                         ; USED BY THE COUNTER. THE SECOND NUMBER IS USED
                         ; TO DETERMINE THE COUNTER TO BE USED.

                         ;DIVIDERS CAN BE FROM 2 TO 0FFFFh, COUNTERS
                         ;  ARE 0,1, & 2
                         ;

  017A    CD 065E        BAUD:   CALL    EXPR           ;GET 2 PARAMS
                                                        ; & PUT THEM ON
                                                        ;  THE STACK
  017D    D1                     POP     D              ;BAUD RATE
  017E    E1                     POP     H              ;COUNTER #
  017F    7D             BRI:    MOV     A,L            ;TEST FOR CTR #
  0180    B7                     ORA     A
  0181    2809                   JRZ     OUTC0
  0183    3D                     DCR     A
  0184    2811                   JRZ     OUTC1
  0186    3D                     DCR     A
  0187    2819                   JRZ     OUTC2
  0189    C2 057F                JNZ     ERROR

  018C    3E36           OUTC0:  MVI     A,00110110B
  018E    D36F                   OUT     CTRTC
  0190    7B                     MOV     A,E
  0191    D36C                   OUT     CTR0
  0193    7A                     MOV     A,D
  0194    D36C                   OUT     CTR0
  0196    C9                     RET

  0197    3E76           OUTC1:  MVI     A,01110110B
```

(Continued on page 90)

Frequency Synthesizer

(Listing continued, text begins on page 80)

Listing One

```
0199    D36F            OUT     CTRTC
019B    7B              MOV     A,E
019C    D36D            OUT     CTR1
019E    7A              MOV     A,D
019F    D36D            OUT     CTR1
01A1    C9              RET

01A2    3EB6    OUTC2:  MVI     A,10110110B
01A4    D36F            OUT     CTRTC
01A6    7B              MOV     A,E
01A7    D36E            OUT     CTR2
01A9    7A              MOV     A,D
01AA    D36E            OUT     CTR2
01AC    C9              RET


;THIS PROGRAM DOES THE POWER ON INITIALIZATION OF
; THE 8253 COUNTER TIMERS. THIS PROGRAM IS CALLED
;  BY THE MAIN INITIALIZE PROGRAM. THE THREE
;   COUNTER TIMERS ARE SET UP FOR BAUD RATE
;    GENERATORS. CHAN. #0 = 4800 BAUD
;      CHAN. #1 = 300 BAUD  CHAN. #2 = 110 BAUD

01C8            CT53IN:
01C8    2E00            MVI     L,0             ;INIT CTR 0
01CA    11 001A         LXI     D,26.           ; FOR 2Mhz CLOCK,
                                                ; 4800 BAUD
01CD    CD 017F         CALL    BRI
01D0    2E01            MVI     L,1             ; CTR 1
01D2    11 01A1         LXI     D,417.          ;  300 BAUD
01D5    CD 017F         CALL    BRI
01D8    2E02            MVI     L,2             ; CTR 2
01DA    11 0470         LXI     D,1136.         ;  110 BAUD
01DD    CD 017F         CALL    BRI
01E0    C9 .            RET
```

End Listing One

Listing Two

```
                      PROGRAM #1

10 X=3                ;TIMER 2 DIVIDE NUMBER
20 Y=3                ;TIMER 0 DIVIDE NUMBER
30 PRINT "INPUT CLOCK FREQUENCY":INPUT C
40 PRINT "      OUTPUT FREQUENCY":INPUT O
50 I=O/C                 ;RATIO OF OUTPUT TO CLOCK
60 PRINT "INPUT ACCURACY IN PPM":INPUT A
70 A=A/1E6
80 Z= (I-(Y/X))/I
90 IF ABS (Z)<A THEN GOTO 1000
100 IF Z>0 THEN 600
110 IF Z<0 THEN 500
120 GOTO 1000
500 X=X+1:GOTO 80
600 Y=Y+1:GOTO 80
1000 PRINT"TIMER #2 DIVIDER" X,"TIMER #0 DIVIDER"Y
1010 PRINT"ACTUAL ACCURACY" Z
1020 GOTO 10
```

End Listing Two

STATEMENT OF OWNERSHIP, MANAGEMENT, AND CIRCULATION

(Act of August 12, 1970, Section 3685, Title 39, United States Code)

1. Title of Publication: Dr. Dobb's Journal, Publication No. 307690.
2. Date of Filing: October 26, 1983.
3. Frequency of Issue: Monthly (12 issues, $25).
4. Location of Known Office of Publication: P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94026.
5. Location of Headquarters or General Business Offices of the Publishers: P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94026.
6. Names and Addresses of Publisher, Editor, and Managing Editor: Publisher, Jane Laidley, P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94026. Editor, Reynold Wiggins, P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94026. Managing Editor, Craig LaGrow, P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94026.
7. Owner: People's Computer Company, P.O. Box E, 1263 El Camino Real, Menlo Park, CA 94026.
8. Known Bondholders, Mortgages, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities: None.
9. For Completion of Nonprofit Organization Authorized to Mail at Special Rates: The purpose, function, and nonprofit status of the organization and the exempt status for Federal income tax purposes have not changed during the past 12 months.
10. Extent and Nature of Circulation:

A. Total number copies printed: Average number copies each issue during the preceding 12 months: 25,106. Actual number copies of single issue published nearest to filing date: 29,580.

B. Paid Circulation. 1. Sales through dealers and carriers, street vendors, and counter sales. Average number copies each issue during preceding 12 months: 6,567. Actual number copies of single issue published nearest to filing date: 11,000. 2. Mail subscriptions. Average number copies each issue during preceding 12 months: 15,842. Actual number copies of single issue published nearest to filing date: 17,280.

C. Total Paid Circulation. Average number copies each issue during preceding 12 months: 22,409. Actual number copies of single issue published nearest to filing date: 28,280.

D. Free distribution by mail, carrier, or other means, samples complimentary, and other free copies. Average number copies each issue during preceding 12 months: 581. Actual number copies of single issue published nearest to filing date: 800.

E. Total distribution. Average number copies each issue during preceding 12 months: 22,990. Actual number copies of single issue published nearest to filing date: 29,080.

F. Copies not distributed. 1. Office use, left over, unaccounted, spoiled after printer. Average number copies each issue during preceding 12 months: 568. Actual number copies of single issue published nearest to filing date: 500. Returns from news agents. Average number copies each issue during preceding 12 months: 664. Actual number copies of single issue published nearest to filing date: 0.

G. Total Average number copies each issue during preceding 12 months: 23,558. Actual number copies of single issue published nearest to filing date: 29,580.

I certify that the statements made by me above are correct and complete.

Jane Nissen Laidley

Jane Nissen Laidley, Publisher

CP/M EXCHANGE

by Robert Blum

The column's a little short this month due to an accelerated deadline caused by CP/M '83. The interface series that I promised would start this month will be held off until next month. I apologize for the delay but I think you will enjoy it once it starts.

User Areas – Bah, Humbug

Several months ago, I cried on your shoulder about how old and decrepit my system had become. Well, since then, new disk drives have been installed, a new motherboard is now keeping a handle on the bus signals, and a 15MB hard disk happily gobbles up all the data I can feed it. Unfortunately, I should never have installed the hard disk because it pointed out to me one black hole that exists in CP/M: there is no easy way to partition logically related groups of data files.

Let me explain a little further. Right from the beginning I delegated my floppy disk drives to specific tasks. Drive A: is called the runtime drive. It contains only executable (.COM) files and any temporary work files that may be needed by the application programs. Drives B: and beyond are used for data files. I adopted this arrangement because it eliminates any confusion when switching from one application system to another. When I want to run word processing, the WP runtime disk goes into drive A:, and one of various WP data disks is mounted in a drive designated for data files. Very simple, very clean, and easy to manage.

Now comes the puzzle: how to allocate the hard disk so it is able to fit into my established scheme. My first attempt was to split the 15MB into two logical CP/M disks. The first one, drive C:, was 8MB in size with all remaining space given to drive D:. Merrily I copied disk after disk of data onto the Winchester. It's hard to describe my feeling of excitement as the data was gobbled up by an insatiable appetite. However, I became concerned after I ran my first directory listing. Displayed was such a hodge-podge of files that I knew something had to be changed.

I shuddered at my first thought – mapping a number of logical floppies onto the hard disk. Fortunately, it was only a fleeting thought. Even as I write this I can hear somebody scraping their fingernails across a blackboard.

User numbers would partition the files the way I wanted, but at the expense of duplicating a number of programs in each of the user areas. The overhead of multiple copies of the same program is mandated by CP/M 2.2's inability to automatically switch user areas when searching for a file. For example, if you are currently in user area three, any file that is opened must also be in the same user area. This includes program overlays. One solution to the problem would be to use a CCP replacement such as ZCPR, but this would only provide a partial solution because an expanded search would only be available when ZCPR was resident. Once the executable program is loaded it has no way of knowing what user area it came from and hence no way of finding its overlays. Unfortunately, when Digital Research designed user areas, they failed to provide a means by which a file's FCB could communicate to the BDOS which user area it is in.

I don't mean to place all the blame on DR for this inadequacy. They provided a BDOS function that will switch the user area for any application program that is smart enough to use it. Unfortunately, none of the application developers decided to use it. Micro Pro took one step in the right direction by modifying WordStar to look at drive A: for its overlay files if they are not found on the currently logged-in floppy. This is better than nothing, but if the user area is not taken into account, it is of little use. Of course, at the time that CP/M 2.2 became available, disk capacities were relatively low and disks were typically swapped for each application. Today, however, hard disks are common and there is no excuse for having to allocate 15MB of space into logical floppies. As much as I hate to admit it, the standard version of CP/M 2.2 simply isn't suited to the hard disk environment.

It wouldn't seem too difficult to trap all the disk I/O BDOS calls and swap the user area for FCBs that contain a file extension of .COM or overlays that reside in the same user area. One problem with this occurs when an overlay file doesn't use .OVR for its extension. However, a little extra code could provide the desired results.

Converting to CP/M Plus would solve my problem. As a standard feature, it will search user area zero for any file that is not found in the current user area, although there are some restrictions. The target file must be marked with the $SYS attribute, and it cannot be written to. This arrangement works well for executable programs, which is what I am complaining about, but for my money they don't get the prize. In designing CP/M Plus, DR decided to add some very useful features. A file can be password-protected, and time and date stamping is available if your system supports a clock of some sort. To store this extra information, extended directory entries are recorded along with the regular directory entries.

Unfortunately, to cross-access user areas requires that the program manually switch the user area by calling the BDOS. I think a little extra work should have gone into allowing a program to pass an extended FCB address to the BDOS when a file is opened. This extended FCB would contain all the additional information necessary to interface to the new features that I have already mentioned, plus the user number. Better yet, instead of a user number, how about using an 8-character text name like the file name?

Making comparisons between different operating systems is usually like comparing apples to oranges, but I want to point out how files are handled on another system that I use daily. To protect the innocent I will call this new system Alpha. Alpha is a Z80-based computer with 64K of memory. The resident operating system requires approximately 10K, and the only transient part of the system is the operator interface, which is equivalent to CP/M's console command processor. The only place that my comparison begins to break down is that Alpha has a separate Z80-based processor that handles all disk I/O. I think my analysis still holds because CP/M Plus requires a minimum of 96K if you want to use the extended features. In Alpha's case, this would also be enough to hold all the disk logic.

A file specification on Alpha is contained in two control blocks. The first is called a "file control block" and is used for various pointers and general parameter passing. A two-byte area of the FCB points to another control block, the "file name block." Contained in the FNB are all the file's attributes in text form. Included are eight characters for the file set name (equivalent to user area), eight characters for the file name, and eight more characters for the file extension. Following this information is a complete description of the file. This includes record length, read and write passwords, and some other control information which isn't germane to this discussion. When the program wants to open a file, the FCB address is loaded into a register and a command code is placed into another. A call is made to the operating system through a restart

instruction, and away you go.

If it sounds simple, it is. Take, for example, the command line necessary to assemble a program:

ASM BLUM-PROGRAM.ASM

A few moments later the BLUM-fileset contains the assembler's output even though I was logged into the TEMP-fileset.

I have worked on a number of computers over the years and Alpha is my favorite. The real surprise comes when one considers Alpha's origin. Until Alpha's operating system was far enough along in the development cycle to stand alone, it was run under CP/M. Even after a few years of refinement, there are a number of CP/M's operational features still being used. Of course, I understand that a lot of things can be done when designing a new system that can't be done when modifying an existing one because of compatibility problems. Still, I think a lot can be accomplished if the motivation exists. I am sure that this discussion won't teach DR anything new; I have met a few of their people and they are sharp. But like so much else in this business, decisions are based on market potential. And as long as the semiconductor manufacturers continue to bring out new and improved processors every couple of years, there is little time left to do anything but convert the existing systems. Well, as I have said before, that's the state of the art.

A Point Well Taken

As I was putting the final touches on this month's column, I received a note from a reader complaining about the apparent lack of proofreading of the twittle subroutine that I ran a couple of months ago. He found the word "asterisk" misspelled in the comments describing the program. I must admit that my first impression was that he was being rather nit-picky, but after a moment's thought (it takes that long for my defense mechanism to readjust), I decided that he had a valid point. To the eyes of many, words are like notes to the ears of a musician. I am not an artist, but I am aggravated by a fuzzy radio station or a scratched record; my reader found a misspelled word to be offensive to his eyes. I apologize for the inexcusable error and will pay more attention to detail in the future.

DRI Patch 3 – SUBMIT.COM

Products Affected: CP/M 2.2

Copyright © Digital Research, Inc. 1982. All rights reserved. Used with permission of Digital Research, Inc. Text adapted from the original.

Error Description: If drive A is not the default drive when you run the SUBMIT program, the $$$.SUB file is created on the currently logged-in disk. Therefore, you cannot run a SUBMIT job from any other drive than A. (This occurs because the CCP will only search drive A for the $$$.SUB file.) After you make the changes shown in Listing One (page 94), the system will always create the $$$.SUB file on drive A.

Patch Procedure: Make a back-up copy of SUBMIT.COM before you use DDT to make the changes in Listing One.

SKIPNIF – Last But Not the Least

Last month David Kirkland shared with us his method of speeding up SUBMIT file truncation. When he submitted those valuable comments he also included his version of the original SKIPIF program written by John Ramsey. Dave's program, SKIPNIF, allows one extra parameter on the command line to specify how many records from the SUBMIT file are to be skipped. This is a valuable addition to an already excellent set of SUBMIT file control routines. I think this brings the subject of SUBMIT file processing to a logical conclusion. By using the many routines that have been published in *DDJ* over the past year, you can achieve a substantial level of control over how SUBMIT files are processed. A complete listing of Dave's SKIPNIF program can be found in Listing Two (page 94).

DDJ

(Listings begin on page 94)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 203.**

CP/M Exchange (Text begins on page 92)

Listing One

```
A>DDT SUBMIT.COM
DDT V2.2
NEXT PC
0600 0100
-D5BB
05BB 00 24 24 24 20   .$$$
05C0 20 20 20 20 53 55 42 00 00 00 1A 1A 1A 1A 1A 1A       SUB...
05D0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A  ..........

-
-S5BB
05BB 00 1
05BC 24 .
-G0
A>SAVE 5 SUBMIT.COM
```

End Listing One

Listing Two

```
;
;  SKIPNIF  -- conditionally skip n commands within a sumbit file
;
;       written by David Kirkland   6/12/83
;       based on Don Wright's QUITIF (DDJ No. 71, Sept. 1982 at 8, 14).
;
;       revised 6/31/83 to incorporate the "zero" option, based on
;       John C. Ramsey's SKIPIF (DDJ No. 82, Aug. 1983 at 100-07).
;
;       Usage:
;
;       SKIPNIF Ambig    fn    n     skip if fn is ambiguous
;       SKIPNIF Exists   fn    n     skip if fn exists
;       SKIPNIF Missing  fn    n     skip if fn does not exist
;       SKIPNIF Null   [ fn ]  n     skip if fn not specified not exist
;       SKIPNIF Zero     fn    n     skip if fn has zero length or does not exist
;
;       Only the first character of the first keyword (the AMBIG or EXISTS or
;       whatever) is checked   If an invalid keyword is entered, the
;       n commands will be unconditionally skipped.
;
;       This code written for Ithaca Intersystems IASM assembler; note
;       that "ccir" is what most people call "cpir", and "ccdr" is "cpdr".
;
tpa     equ     0100h
bdos    equ     0005h
fcb1    equ     005ch               ; first fcb prepared by CCP
name1   equ     fcb1+1              ; start of filename of fcb1
fcb2    equ     006ch
name2   equ     fcb2+1
cmdtail equ     0080h
;
drive   equ     'M'                 ; drive on which $$$.SUB file resides
;                                   ; ('A' for normal folk)
;  BDOS codes
```

(Continued on page 96)

CP/M Exchange (Listing continued, text begins on page 92)

(Listing continued, text begins on page 92)

Listing Two

```
;
PrtStr  equ     9
Open    equ     15
Close   equ     16
Srchlst equ     17
Erase   equ     19
Read    equ     20
;
skipif  org     tpa
;
; first parse command line by
; - determining NSkip, the number of lines to skip if condition satisfed
; - ensuring that the contents of fcb2 at 6c contain the fn argument and
;   not the NSkip argument (this problem occurs when the command is
;       skipnif null NSkip
;   i.e., when the fn argument actually is not entered and the skip should
;   be executed).
;
        lxi     h,cmdtail
        mov     c,m                     ; length of command line
        mvi     b,0
        dad     b                       ; hl points to last character in command line
        inx     h
        mvi     m,' '                   ; mark end of command line with space
; skip over "NSkip", the last argument on command line
        call    scanbl                  ; scan off trailing blanks
```

```
        ccdr                            ; scan backwards until a space reached
        push    h                       ; save for later

; now skip back over the next to last arg, which is "fn" if present
        call    scanbl                  ; find end of next arg
        ccdr                            ; find start of the arg

; now move back once more.  Normally, there will be one argument left, the
; AMBIG or NULL or whatever.  But in our special case of NULL with no
; fn entered, there will be nothing left.
        call    scanbl                  ; throw away extra blanks
        mov     a,b                     ; are there any characters unprocessed in the
        ora     c                       ; command line? if not, then there were only
        jnz     DoNum                   ; two arguments given on the command line

match   mvi     a,' '                   ; if only two args, add a blank for the second
        sta     name2                   ; arg to make fcb2 null
        jmp     DoNum

; scanbl scans backwards over the command line discarding blanks;
; when a nonblank character is encountered, it stops.
; h -> str, bc = length remaining; return with hl -> first nonblank

scanbl  mov     a,b                     ; see if any characters remain
        ora     c
        rz                              ; none left
        mvi     a,' '
scnLp   cmp     m                       ; *hl : ' '
        rnz
        dcx     h                       ; next char
        dcx     b                       ; count down
```

(Continued on next page)

CP/M Exchange (Listing continued, text begins on page 92)

Listing Two

```
        jmp     scanbl

;
; PROCESS DECIMAL NUMBER--modified from ZCPR
;
DoNum
        pop     h
        inx     h
        inx     h                   ; hl points to first char in number
; convert the decimal number pointed to by hl into binary and store
; at N2Skip
        mvi     c,0                 ;C=ACCUMULATED VALUE set to zero
NUM1:
        MOV     A,M                 ;GET CHAR
        CPI     ' '                 ;DONE IF ' ', which marks end of number
        JZ      NUM2
        INX     H                   ;PT TO NEXT CHAR
        SUI     '0'                 ;CONVERT TO BINARY (ASCII 0-9 TO BINARY)
        CPI     10                  ;ERROR IF >= 10
        JNC     NUMERR
        MOV     D,A                 ;DIGIT IN D
        MOV     A,C                 ;NEW VALUE = OLD VALUE * 10
        RLC
        RLC

        RLC
        ADD     C                   ;CHECK FOR RANGE ERROR
        JC      NUMERR
        ADD     C                   ;CHECK FOR RANGE ERROR
        JC      NUMERR
        ADD     D                   ;NEW VALUE = OLD VALUE * 10 + DIGIT
        JC      NUMERR              ;CHECK FOR RANGE ERROR
        MOV     C,A                 ;SET NEW VALUE
        jmp     NUM1                ;do next digit
;
;
NUMERR:                             ; come here if number invalid
        mvi     c,PrtStr
        lxi     d,NErrMsg
        call    bdos
        ret
;
                                    ; come here once number finished
NUM2:
        MOV     A,C                 ;GET ACCUMULATED VALUE
        sta     N2Skip              ; store the value for later

; now actually do what the caller asked for:

        lda     name1               ; first character of the AMBIG or EXISTS or ...
        lxi     h,name2

        cpi     'A'
        jz      ambig
        cpi     'E'
        jz      exists
        cpi     'M'
        jz      missing
        cpi     'N'
        jz      null
```

(Continued on page 100)

CP/M Exchange (Listing continued, text begins on page 92)

(Listing continued, text begins on page 92)

Listing Two

```
        cpi     'Z'
        jz      zero
                                ; if none of the above, fall through and skip

; skip over next N2Skip commands in $$$.SUB
skip    mvi     c,Open          ; open $$$.SUB
        lxi     d,subfile
        call    bdos
        inr     a               ; check if $$$.SUB exists
        rz                      ; if not, return

        lxi     h,subfile+14
        mvi     m,0             ; zero the S2 byte of the fcb to force a write
                                ; of the modified fcb back to the directory
        lda     N2Skip
        mov     c,a
        inx     h
        mov     a,m             ; decrement the number of records in $$$.SUB
        sub     c               ; by N2Skip
        jp      notneg
        xra     a               ; if skip by more than in file, use zero
notneg  mov     m,a

        mvi     c,Close         ; close $$$.SUB to rewrite the modified fcb
        lxi     d,subfile
        call    bdos

        mvi     c,PrtStr        ; tell the user what we've done
        lxi     d,skipmsg
        call    bdos

        xra     a               ; zero a and set flags for zero/missing
        ret

; abort if 2d parm ambiguous
ambig   mvi     a,'?'
        lxi     b,11
        ccir                    ; cpir for people with normal assemblers
        jz      skip
        ret

; abort if file exists
exists  call    search
        jnz     skip
        ret

; abort if file missing
missing call    search
        rnz
        jmp     skip

; abort if 2d parm null
null!   mvi     a,' '
        cmp     m
        rnz
        lxi     h,name2+8
        cmp     m
        rnz
        lda     fcb2
        ora     a
```

```
        rnz                              ; d: given without more
        jmp     skip

; abort if 2d parm is a zero length file
zero    call    missing                  ; if file does not exist, its deemed zero length
        rz                               ; if so, missing has skiped N2Skip records in
                                         ; $$$.SUB, so now we return to the CCP
        mvi     c,Open
        lxi     d,fcb2
        call    bdos                     ; open the specified file

        mvi     c,Read
        lxi     d,fcb2
        call    bdos                     ; try to read the first record.
        ora     a
        rz                               ; read was OK, so not zero length, don't skip.
        jmp     skip                     ; read failed, zero length.
; search for first to see if any file exists to match fcb
search  mvi     c,Srch1st
        lxi     d,fcb2
        call    bdos
        inr     a                        ; adjust flags: Z <-> not found
        ret
;
N2Skip  db      0

skipmsg db      "Next commands skipped",13,10,'$'
NErrMsg db      "Illegal number to skip",13,10,'$'
subfile db      drive+1-'A',"$$$      SUB",0,0,0,0
;
```

End Listing

by Ray Duncan

Version 2.0 of MS-DOS was enhanced with some 40 new function calls for use by application programs. These provide support for the hierarchical directories, device-independent I/O, and memory allocation. Most of them are rather radical departures from the CP/M-ish origins of MS-DOS and show a clear tendency toward providing a Unix-like environment.

In the process of preparing a PC-DOS programmer's reference book for publication next year, I have been trying each of the new functions out in turn by writing small assembly language utilities. When I reached function 4BH, called EXEC, I was especially intrigued. EXEC is designed to allow an application program (which we may call the "parent") to load and execute any other program from the disk, then regain control when the "child" is finished. The parent can pass a considerable amount of information to the child in the form of a command line, default File Control Blocks, and a set of strings called the "environment." Similarly, the child can pass an exit code back to the parent when it finishes its work.

The child program can in turn load other programs, and so on through many levels of control until you run out of memory. This function has plenty of uses even in a single-tasking, single-user operating system. For example, you could write your own screen-oriented, menu-driven command processor which would become the user's interface – one would never need to learn the more complex commands like MODE, CHDIR, and PATH (the same thing could be accomplished with a batch file but it would run much slower). Alternatively, you could load a second copy of the native Command Line Processor (COMMAND.COM), and let it run under the control of your program, rather than directly on top of MS-DOS.

In a multi-tasking environment such as we expect to see in MS-DOS 3.0, this function will be even more useful and will undoubtedly be elaborated with several additional features. Under such an operating system, a parent task can "spawn" any number of child tasks, which can execute concurrently and asynchronously, and communicate by means of queues, semaphores, and pipes.

Well, you say, pie in the sky is all very nice, but why is the EXEC function getting so much attention in this magazine column? The answer is, of course, that when I tried to actually use the function I ran into any number of glitches and hazy spots in the documentation.

The "official" definition of the EXEC function, slightly boiled down, is given in Figure 1 (below). It looks straightforward enough, if you ignore the problem of what format the parameter block's DWORD addresses might be in. The documentation notes that "when your program received control, all available memory was allocated to it. You must free some memory (see call 4AH) before EXEC can load the program you are invoking. Normally, you would shrink down to the minimum amount of memory you need, and free the rest."

The manual goes on to state that all open files of the parent are duplicated in the newly created child task. Also inherited from the parent is some new creature called an "environment block," which is rather vaguely defined as a series of ASCII strings in the form "NAME=parameter."

Each of the strings in the environment block is terminated by a zero byte, and the whole set of strings is terminated by an additional zero byte. The environment always starts on a paragraph boundary, and its segment address is ultimately placed at offset 2CH of the Program Segment Prefix. You can either leave that slot in your EXEC parameter block as zero (in which case the child will acquire the

Call with:

AH	=	4BH
AL	=	load type (see below)
DS	=	segment of ASCIIZ path & program name string
DX	=	offset of ASCIIZ path & program name string
ES	=	segment of parameter block
BX	=	offset of parameter block

Returns:

Carry Flag set if unsuccessful call. AL = error code.

Carry Flag cleared if successful call. Except for CS and IP, all other registers, including the stack pointers, are destroyed.

Load type = 0 : Load and execute program. "A program segment prefix is established for the program and the terminate and control/Break addresses are set to the instruction after the EXEC call."

Parameter block format:

WORD	segment address of environment block
DWORD	pointer to command line to be placed at new Program Segment Prefix + 80H
DWORD	pointer to default FCB #1, to be passed at new Program Segment Prefix + 5CH
DWORD	pointer to default FCB #2, to be passed at new Program Segment Prefix + 6 CH

Load type = 3 : Load overlay. Does not create a Program Segment Prefix, and does not begin execution of the overlay.

Parameter block format:

WORD	segment address where file will be loaded
WORD	relocation factor to be applied to the image

Figure 1.

Specification of arguments and results of MS-DOS EXEC function 4BH.

same environment block as your calling program), or you can provide a pointer to your own set of strings. The manual nonchalantly adds that the maximum size of the environment block is 32 Kbytes.

This zinger introduces a whole new set of mysteries. What in the world could the environment block possibly contain that could be as large as 32 Kbytes? Furthermore, the manual explicity states that the environment block for any given program is static, implying that if more than one generation of child programs is resident in RAM, each one might have a distinct and separate copy of these potentially 32 Kbyte rascals floating around somewhere. My goodness!

Paging through the PC-DOS manual in search of further enlightenment, I found some help in appendix E which stated: "the environment built by the command processor (and passed to all programs it invokes) will contain a COMSPEC string at a minimum (the parameter on COMSPEC is the path used by DOS to locate COMMAND.COM on disk). The last PATH and PROMPT commands will also be in the environment, along with any environment strings entered through the SET command."

Thus, the environment block contains at most three strings that actually mean something to MS-DOS (PATH=, COMSPEC=, and PROMPT=). Any other strings found in the block by an application program were placed there for its information by a SET command issued by the operator or by another application program and have no effect on the behavior of the operating system proper. Loading a dummy COM file under the control of DEBUG, getting the pointer from offset 2CH in the Program Segment Prefix, and dumping out memory, I found that the "default" environment in my system is indeed only the two strings "PATH=" and "COMSPEC=A:\COMMAND.COM." Not so formidable after all.

The major remaining puzzle turned out to be the SETBLOCK function (MS-DOS function 4AH), which your program is supposed to use to "shrink down" the amount of space allocated to it before using EXEC to load another program. The manual says, "on entry, ES contains the segment of the block, BX contains the new requested block size in paragraphs." But when the "block" refers to the application program itself, and not to some piece of RAM it requested later, what segment to use: the segment of the Program Segment Prefix or the base of the Code Segment? Or does "block" actually refer to some kind of memory descriptor parameter block buried in COMMAND.COM?

Attempts to test this out with the Debugger were completely baffling, every attempt resulting in an error code. It turns out that since the Debugger essentially "owns" all of memory and loads your

program via its own mechanisms, a SETBLOCK call by a program running under the control of DEBUG is illegal by definition – from the operating system's point of view, it is trying to modify a memory allocation that does not "belong" to it. In any event, the "segment of the block" that should be fed to function 4AH is the Program Segment Prefix.

Finding the number of paragraphs to "shrink" your program's memory allocation down to is also a little tricky, at least for EXE files. You might think that you can simply make an Equate to an address called Program_End, put that in a data segment at the end of your program, and load that segment address into BX at execution time with the aid of the GROUP modifier, thereby letting the Linker and Loader do all the work.

Unfortunately, that method won't work – at least not all the time. The Linker moves all the declared segments around to suit its own purposes, and the physical order of segments in your source code doesn't usually correspond to the order of segments in the resultant EXE file. The best way I have figured out to get the total size of the program at runtime is to put an equate to "Program_End" into another module, which is separately assembled, and reference it with an EXTRN in your main program. At link time, specify the module containing the "Program_End" equate as the last in the list of OBJ files.

Of course, you can also "cheat" by simply assembling and linking the program, looking at the size of the EXE file in the directory listing, and then reassembling the program with the correct size in the code as a literal value. This assumes that all data areas used by the program are embedded as declared constants and variables. An approach like this could cause you trouble later if you modify the program in a way that increases its size, then forget to update the literal too.

To keep the amount of painful interaction between the EXEC function and *DDJ* readers to a minimum, I am publishing herewith a very detailed example program named FXN4BH (see the listing, below), using EXEC to load the MS-DOS utility CHKDSK.COM, that is guaranteed to work. It requires that both FXN4BH.EXE and CHKDSK.COM be present on the "current" or default disk drive. FXN4BH can fail if there is not enough room above it in RAM to load and execute CHKDSK.COM, or if the transient portion of COMMAND.COM in highest RAM (which contains the actual loader) has been destroyed and there is not enough free memory to reload it.

DDJ

(Listing begins below)

16-Bit Toolbox Listing

(Text begins on page 102)

```
                                        name    FXN4BH
                                        page    55,132
                                        title   'FXN4BH --- demo PC-DOS EXEC function'
                                ;
                                ; FXN4BH --- demonstrate use of the
                                ; PC-DOS 2.0 EXEC function call 4BH
                                ;
                                ; Copyright (c) 1983 by Ray Duncan
                                ;
      = 000D                    cr      equ     0dh         ;ASCII carriage return
      = 000A                    lf      equ     0ah         ;ASCII line feed
                                                            ;
      0000                      cseg    segment para public 'CODE'
                                                            ;
                                        assume  cs:cseg,ds:data,ss:stack
                                                            ;
      0000                      demo    proc    far
                                                            ;at entry DS & ES = PSP
      0000  1E                          push    ds          ;Save address for final
      0001  33 C0                       xor     ax,ax       ;FAR RET to PC-DOS on stack
      0003  50                          push    ax
                                                            ;save copy of SS:SP for use
                                                            ;after return from overlay
      0004  2E: 8C 16 0050 R            mov     cs:STK_SEG,ss
      0009  2E: 89 26 0052 R            mov     cs:STK_PTR,sp
                                                            ;
                                                            ;Reserve 1000H bytes for
                                                            ;this loader and release
                                                            ;the rest of memory for
                                                            ;use by the overlayed program.
      000E  BB 0100                     mov     bx,100h     ;ES=segment of PSP of loader
```

(Continued on page 106)

```
0011  B4 4A                 mov     ah,4ah    ;BX=paragraphs to reserve
0013  CD 21                 int     21h
                                              ;make the messages in data
                                              ;segment addressable
0015  B8  ---- R            mov     ax,seg DATA
0018  8E D8                 mov     ds,ax
001A  8E C0                 mov     es,ax
                                              ;jump if memory
                                              ;de-allocation failed
001C  72 2A                 jc      ALLOC_ERR
                                              ;print memory successfully
                                              ;released
001E  BA 001E R             mov     dx,offset MSG2
0021  B4 09                 mov     ah,9
0023  CD 21                 int     21h
                                              ;
                                              ;now load and execute
                                              ;the overlaid program.
0025  BA 00A4 R             mov     dx,offset PGM_NAME
0028  BB 00B0 R             mov     bx,offset PAR_BLK
002B  B0 00                 mov     al,0
002D  B4 4B                 mov     ah,4bh
002F  CD 21                 int     21h
                                              ;restore stack pointers
                                              ;to state before EXEC call
0031  2E: 8E 16 0050 R      mov     ss,cs:STK_SEG
0036  2E: 8B 26 0052 R      mov     sp,cs:STK_PTR
                                              ;Make data segment
                                              ;addressable again
003B  B8  ---- R            mov     ax,seg DATA
003E  8E D8                 mov     ds,ax
                                              ;print message that loader
                                              ;successfully regained control
0040  BA 0059 R             mov     dx,offset MSG3
0043  B4 09                 mov     ah,9
0045  CD 21                 int     21h
                                              ;now exit to PC-DOS
0047  CB                    ret

0048             alloc_err:                   ;come here if memory
                                              ;cannot be released
0048  BA 0000 R             mov     dx,offset MSG1
004B  B4 09                 mov     ah,9
004D  CD 21                 int     21h       ;print error message and
004F  CB                    ret               ;exit to PC-DOS
                                              ;
0050             demo       endp
                                              ;
                                              ;these two variables must
                                              ;reside in Code Segment so
                                              ;that they are addressable
                                              ;after return from overlay.
```

(Continued on page 108)

```
     0050 0000            stk_seg  dw      0          ;original SS contents
     0052 0000            stk_ptr  dw      0          ;original SP contents
                                                      ;
     0054                 cseg     ends

                                                      ;declare a stack area
                                                      ;for use by this loader
     0000                 stack    segment para stack 'STACK'
                                                      ;allow 64 bytes in this case
     0000     40 [                 db      64 dup (?)
                    ??
                         ]

     0040                 stack    ends

                                                      ;declare data segment to
                                                      ;contain variables and tables
     0000                 data     segment para public 'DATA'
                          ;
     0000 0D 0A           msg1     db      cr,lf
     0002 55 6E 61 62 6C 65        db      'Unable to release memory.'
          20 74 6F 20 72 65
```

```
              6C 65 61 73 65 20
              6D 65 6D 6F 72 79
              2E
      001B    0D 0A 24                      db      cr,lf,'$'
      001E    0D 0A              msg2       db      cr,lf
      0020    4D 65 6D 6F 72 79             db      'Memory above loader released.'
              20 61 62 6F 76 65
              20 6C 6F 61 64 65
              72 20 72 65 6C 65
              61 73 65 64 2E
      003D    0D 0A 4E 6F 77 20             db      cr,lf,'Now loading CHKDSK.COM.'
              6C 6F 61 64 69 6E
              67 20 43 48 4B 44
              53 4B 2E 43 4F 4D
              2E
      0056    0D 0A 24                      db      cr,lf,'$'
      0059    0D 0A              msg3       db      cr,lf
      005B    4C 6F 61 64 65 72             db      'Loader regained control from CHKDSK,'
              20 72 65 67 61 69
              6E 65 64 20 63 6F
              6E 74 72 6F 6C 20
              66 72 6F 6D 20 43
              48 4B 44 53 4B 2C
      007F    0D 0A                         db      cr,lf
      0081    6E 6F 77 20 6D 61             db      'now making final exit to PC-DOS.'
              6B 69 6E 67 20 66
```

(Continued on next page)

16-Bit Toolbox Listing (Listing continued, text begins on page 102)

```
        69 6E 61 6C 20 65
        78 69 74 20 74 6F
        20 50 43 2D 44 4F
        53 2E
 00A1   0D 0A 24                        db      cr,lf,'$'
                               ;
                                                 ;drive, path, and name of program
                                                 ;to be loaded and executed.
 00A4   5C 43 48 4B 44 53      pgm_name db      '\CHKDSK.COM',0
        4B 2E 43 4F 4D 00
                               ;
 00B0   ---- R                 par_blk  dw      ENVIR    ;segment address of
                                                 ;environment descriptor
                                                 ;
                                                 ;full address of command line
                                                 ;to be passed at offset 80H
 00B2   00BE R                          dw      offset CMD_LINE    ;in overlaid
 00B4   ---- R                          dw      seg CMD_LINE       ;program's PSP
                                                 ;
                                                 ;full address of default
                                                 ;File Control Block to be
                                                 ;passed at offset 5CH in
 00B6   00C5 R                          dw      offset FCB1        ;overlaid
 00B8   ---- R                          dw      seg FCB1           ;program's PSP
                                                 ;
                                                 ;full address of default
                                                 ;File Control Block to be
                                                 ;passed at offset 6CH in
 00BA   00EA R                          dw      offset FCB2        ;overlaid
 00BC   ---- R                          dw      seg FCB2           ;program's PSP
                               ;
                                                 ;actual command line tail
                                                 ;to be passed to overlay
 00BE   04 20 2A 2E 2A 0D      cmd_line db      4,' *.*',cr,0
        00
                               ;
                                                 ;first default FCB to
 00C5   00                     fcb1     db      0        ;be passed to overlay
 00C6      0B [                         db      11 dup ('?')
                3F
                   ]

 00D1      19 [                         db      25 dup (0)
                00
                   ]


                                                 ;second default FCB to
 00EA   00                     fcb2     db      0        ;be passed to overlay
 00EB      0B [                         db      11 dup (' ')
                20
                   ]

```

```
00F6     19 [                     db      25 dup (0)
                  00
                     ]

                              ;
010F                          data    ends

                                                        ;declare separate data
                                                        ;segment to contain
                                                        ;environment descriptor
0000                          envir   segment  para 'ENVIR'
                                                        ;
                                                        ;Search path used by PC-DOS
                                                        ;to look for commands or
                                                        ;batch files not found in
0000  50 41 54 48 3D 00               db       'PATH=',0 ;the current directory
                                                        ;
                                                        ;Search path used by PC-DOS
                                                        ;to locate COMMAND.COM
0006  43 4F 4D 53 50 45               db       'COMSPEC=A:\COMMAND.COM',0
      43 3D 41 3A 5C 43
      4F 4D 4D 41 4E 44
      2E 43 4F 4D 00
001D  00                              db       0        ;extra 0 byte designates
                                                        ;end of environment
001E                          envir   ends

                                      end      demo
```

End Listing

TRANSFORM

Company: MasterComputing, Inc., P.O. Box 17442, Greenville, SC 29606
Price: $39.95
Circle Reader Service No. 121
Reviewed by Michael Favitta

MasterComputing's (MCI) TRANSFORM is a language preprocessor for Microsoft BASIC and other compatible BASIC's. It translates MCI's enhanced BASIC syntax (which includes block structured statements) into Microsoft BASIC 5.X syntax. The purpose of TRANSFORM is to allow BASIC programmers to develop programs that are easier to debug, understand, and maintain, by using a structured source language.

TRANSFORM is designed for CP/M systems that have 48K of memory. Translated programs will run under MBASIC 5.X or can be compiled with BASCOM 5.3. The translator is distributed on an 8-inch single-density disk along with a sample program and a utility program named SIFT.

A Structured BASIC Language

MCI has made three changes to Microsoft BASIC to create an enhanced version of BASIC. MCI has added blocked control structures, an INCLUDE facility, and replaced line numbers with labels.

To add blocked structures to BASIC, the Microsoft BASIC statements ON-GOTO and IF-THEN-ELSE were modified and a new statement, REPEAT-UNTIL, was added. The syntax of all other statements remains the same. The end-of-block sentinels, ENDGOTO and ENDIF, were added to the standard syntax of the ON-GOTO and IF-THEN-ELSE statements. The translator implements the three additional statements by replacing MCI's blocked structures with combinations of standard Microsoft IF-THEN and GOTO statements. This produces an intermediate object program that is standard Microsoft BASIC. The two examples in Table I (at right) show how the IF-THEN-ELSE-ENDIF and REPEAT-UNTIL statements are translated.

Line numbers are replaced with labels in the TRANSFORM source language. A label is any combination of alphanumeric characters up to 255 characters in length that is preceded by the character "@". The translator generates the line numbers that are required by Microsoft BASIC when the intermediate object file is created. A symbol table can be produced that contains the label to line number mapping. A restriction: only comments may follow a label definition, since the translator turns all label definitions into REMarks for documentation in the intermediate object program.

MCI added an INCLUDE facility to allow any file that contains BASIC statements to be copied into a program. A DEFINE statement may be used in conjunction with an INCLUDE statement to redefine the variable names in the included source. This allows a programmer to use the same procedure in many programs, or several times in a single program, without modifying the variables used in the procedure. Since all BASIC variables are global, using a facility such as MCI's INCLUDE is as close as you can get to having local procedures in BASIC. The INCLUDE statement may be nested up to five levels deep.

TRANSFORM allows a programmer to enable and disable the translator from within the program. Two special labels, @PREON and @PREOFF, are used for this purpose. When the translator is disabled, the standard Microsoft BASIC ON-GOTO and IF-THEN-ELSE statements can be used. @PREON and @PREOFF function correctly with the INCLUDE statement by only affecting the level of INCLUDE at which they were specified.

The TRANSFORM Translator

The TRANSFORM translator consists of two COM files, TF.COM and TF1.COM. To invoke the translator you need only enter TF. The program prompts the user for all information that is needed to perform the translation. Most options have a default response that can be chosen by entering a carriage return. Default file name extensions of PRE and BAS are assumed for the source input and object files. TRANSFORM searches for the source if it is not on the specified drive. The order of the search is the currently logged-in drive, followed by the system drive. TRANSFORM's processing options include line number and increment selection for the translated source, listing and error message routing, hardcopy formatting information, and symbol table creation.

Once file name and option selection is complete, TRANSFORM begins its first pass on the source file. This pass translates the structured BASIC source program to standard Microsoft BASIC. The

TRANSFORM Source	Translated Object Code
IF (condition) THEN	1000 IF NOT (condition) THEN 1200
.	.
(block of statements)	(block of statements)
.	.
ELSE	1190 GOTO 1400
	1200 REM
.	.
(block of statements)	(block of statements)
.	.
ENDIF	1400 REM
REPEAT	1000 REM
.	.
(block of statements)	(block of statements)
.	.
UNTIL (condition)	1200 IF NOT (condition) THEN 1000

Table 1
Examples of how the IF-THEN-ELSE-ENDIF and REPEAT-UNTIL are translated.

translator may be interrupted at any time. Once interrupted, you may continue, translate a different source file, or terminate the translator. At the completion of pass one, TRF.COM is loaded and pass two begins. This pass performs all the line numbering that is required by Microsoft BASIC. When the translator has finished, an intermediate object program has been created which can be loaded by MBASIC or compiled by BASCOM.

TRANSFORM programs must be created with a text editor, since BASIC editors always require line numbers. Some text editors, such as Wordstar, set the high order bit on certain characters. This causes problems for the translator, so a utility named SIFT is provided to strip off all the high order bits in a text file. This utility is not needed if your text editor doesn't set the high order bit on any ASCII characters.

Documentation

The documentation that is provided by MCI is divided into two sections, a tutorial and a user's manual. The tutorial demonstrates how the translator is invoked and describes the procedures that are needed to get from MCI-enhanced BASIC to Microsoft BASIC. An additional feature in the tutorial is a discussion of structured programming and how it can be used to simplify the task of creating an "error-free" program. As an example, the tutorial proceeds step by step through the design of a program to format and print BASIC program source files. The source code for the example program is included, so that the user may experiment with TRANSFORM without first having to write a program. Besides being the basis for the tutorial, the print program is a useful utility.

The user's manual is a logical progression of information about TRANSFORM. The first part of the manual contains general information on the purpose of TRANSFORM and its theory of operation. The second part details the syntax of all aspects of MCI's enhanced BASIC language.

Reviewer's Comments

TRANSFORM allows a programmer to write BASIC programs using blocked control structures that have been present in other high-level languages for years. Ideally, there would be no use for such a product, as people who wished to write structured programs would use an available high-level language. Unfortunately, due to the number of years Microsoft BASIC has been around, the availability of a compatible compiler, and the costs associated with purchasing several language processors, this product does serve a useful purpose. Many CP/M software products also require Microsoft BASIC, such as MCI's MCDISPLAY, making the use of a structured language impossible.

One disadvantage to MCI's approach to structured BASIC is that it adds another step in the process of creating an executable program. It is a costly step regarding compile time, as translating a 500-line program takes between eight and ten minutes. There is also potential for problems in the translated source due to the way TRANSFORM implements its blocked structures. In Microsoft BASIC any non-zero expression is considered to represent the true condition for an IF statement. MCI uses "IF NOT(condition) THEN" as part of the translation for each of the blocked structures. This creates the restriction that all conditionals used by the blocked control structures must evaluate to true (minus one) or false (zero). Although this restriction is well-documented, the burden is on the user to make sure that it is followed.

MCI's product TRANSFORM does work. The documentation is complete and the translator is easy to use. If you already have Microsoft BASIC, it would be difficult to find a structured language interpreter or compiler of this caliber for $39.95.

COMMX-PAC

Company: Hawkeye Grafix, 23914 Mobile, Canoga Park, CA 91307
Computer System: CP/M 2.0 or greater
Price: COMMX-PAC: $150, or COMMX only: $99
Circle Reader Service No. 123
Reviewed by Gene Head

I write for several computer magazines and have a listed phone number. So I suppose I get more than my normal share of phone calls from individuals with questions about microcomputers. Eight out of ten callers are requesting help getting a modem working. Seven of those eight are non-programming types trying to get some version of Ward Christensen's MODEM program working. For many, it is a simple matter to set a program's equates to a serial RS-232 status and data port and to monitor the proper status bits For just as many this task is impossible.

Will Pierce at Hawkeye Grafix has taken on the task of delivering a modem communications and control package that is factory patched to run without change on the user's computer system. When ordering COMMX-PAC, the user specifies his/her computer and modem, and Hawkeye does the rest. The purchaser receives a disk with programs that run without the need for any adaptations. Well almost. . . .

The package I ordered for my IMSAI MIO serial I/O card was delivered with the status bit logic reversed. In other words, when executing a status check, the instruction JZ (jump if zero) should have been JNZ (jump if not zero). I know the MIO card is working properly and wired per the manufacturer's specifications because the firmware monitor in a factory ROM controls the MIO card without a hitch. Now it should be noted that IMSAI has been out of business for several years and expecting a vendor to support equipment which is no longer supported by even the manufacturer could be asking too much.

The package my company ordered for a brand-new Molecular Super Micro 30 had two fatal bugs. Molecular's N/Star operating system uses an expanded and larger-than-normal CCP. Fortunately, this package was purchased as assembly language source code. I had no trouble tracking down three places in the source code that expected the normal sized CCP and making the proper modifications. The second error surfaced when the "phone is ringing" response from the smart modem caused the activity time-out timer to reset. This was fixed by telling the smart modem not to send responses after a caller connected.

COMMX-PAC includes, in addition to COMMX and CONSOLX (discussed below), a "data base management" program and a message board program both written in Hawkeye Grafix compiled VBASIC. Neither of these programs performs as well as the public domain programs available on larger RCPM systems and through user's groups.

COMMX-PAC is two separate packages: COMMX for connecting to a host computer like the Source or an RCP/M and CONSOLX for making your computer act like a host or RCP/M.

COMMX

COMMX is the terminal program that allows your computer to communicate with other (host) computers. COMMX is far superior to even the most advanced public domain MODEM program that I have seen to date. It can virtually communicate with any system like IBM TSO, HP-3000, as well as RCP/M's. It has keyboard macro definitions; an entire call, connect, and log-on procedure can be compressed to one or two keystrokes. A half-dozen separate programs allow for an automatic dial-up directory, encoding and decoding private files and un-loading binary files to INTEL HEX format files for systems that are restricted to sending and receiving only seven bits of data.

Hawkeye has included Dick Greenlaw's squeeze and un-squeeze programs among its utilities. These can cut modem connect time by 50% by compacting the

file to be sent, sending the compacted version and then un-compacting it at the receiving end. There is also a sorted disk directory utility but this utility is not as impressive as the public domain program SD-44.COM (Super Directory) described in The CP/M Exchange column of *DDJ*, June 1983.

COMMX supports two error-checking protocols and two methods of file transfer. The COMMX-to-COMMX file transfer system seems to be more advanced than the COMMX-to-MODEM7 system. Earlier versions of COMMX didn't support the Christensen protocol and I'm glad to see the newer version does.

The most advanced feature of COMMX is the E-MAIL feature. Using the dial-up directory, keyboard macros, and a submit file, COMMX can establish communication with other COMMX/CONSOLX sites and exchange mail files while each site is unattended. The entire process can be initiated and then delayed a specified number of hours and minutes to take advantage of off-hour phone charges. COMMX continues trying to establish communications until the job is completed. This E-MAIL requires the most effort to get operational, but the manual is clear and well laid out in describing the E-MAIL features.

The manual also includes an index and even a glossary for non-computing types to refer to. Eighty percent of the manual is dedicated to COMMX with the remaining twenty percent devoted to CONSOLX.

CONSOLX

CONSOLX, like COMMX, is configured at the factory to the user's machine and modem. The list of supported machines and modems is long and growing. I guess nine of ten personal as well as business micros are currently supported.

CONSOLX allows your computer to function as a host to callers accessing your system. Field selectable options include a password, ring-back operation, a welcome message, and UART configuration to either 7-bit even parity or 8-bit no parity. An auto start feature allows your system to execute a program after the user gets past the optional password and welcome message.

At the Money Tree (the company I work for) we run a system that logs callers in, gives them access to proprietary financial information, and logs them off, keeping track of time and charges. Because of its ability to answer the phone and then autoload and execute other programs, CONSOLX works well as the base system. Callers see only as much of the system as they need to see. I know of at least one other business using CONSOLX commercially.

Conclusion

I had to write the routines to switch baud rates in the CONSOLX program and I think these should be included in the package. Also, working without carrier detect requires disorderly log-offs to time out for five minutes before the next caller can connect. I was able to get around this on the Molecular, but I'm not sure this problem can be solved generally for all systems.

For some of us, getting a program to work is half the fun of computing. For others who want it to work the first time and every time, COMMX-PAC from Hawkeye Grafix is clearly worth the $150 price tag. Furthermore, if you have no need for making your computer a host computer and only want to originate calls to other host systems, the COMMX program alone sells for $99.

I commend Hawkeye Grafix for an outstanding package in COMMX-PAC. They have taken on a tough job of providing software for various hardware environments, and as far as I know they are doing the job well. My calls to Will Pierce have always been given professional yet personal treatment. I can personally recommend all their fine products, especially the COMMX-PAC communications package.

DDJ

An Introduction to Programming and Problem Solving with Pascal, Second Edition

By G. Michael Schneider, Steven W. Weingart, and David Perlman
Published by John Wiley & Sons, Inc.
$21.95, 468 pages
Reviewed by William G. Hutchison, Jr.

"What? *Another* Pascal textbook?" If your reaction was the same as this reviewer's, please do not give up in despair. This book is not another quick-and-dirty attempt to cash in on the demand for computer texts; it is a thoroughly professional effort. It comes from that hotbed of Pascal activity, the University of Minnesota. Andy Mickel read the proof copy. You may remember him as the former editor of *Pascal News.* Niklaus Wirth was interested enough in the project to write a foreword.

This is clearly a college text. The authors wrote it to be used as text for the course CS I, Computer Programming I, of ACM Curriculum '78. They have achieved a very good balance of subject material. Some authors overload their readers with thorny calculus problems to be programmed. This is useful material, but not suitable for an introductory programming course. This text is evenly balanced between text processing applications, commercial (payroll and inventory), games (LIFE), and elementary statistics. The authors use real programs as examples, not meaningless fragments cooked up just to display some feature of the language.

The authors follow their own advice, which is to plan what to do before beginning the coding. Chapter 1 is devoted to problem specification. They point out correctly that most program errors start from errors or ambiguities in the statement of the problem, so it is worthwhile to spend considerable time getting it right at the beginning. Chapter 2 is about developing algorithms in pseudo-code. It contains a brief discussion of program efficiency and asymptotic analysis (e.g., big-O function, n-squared versus n log n performance), not enough to bog down the reader, but enough to stimulate the student to take the next course.

In addition to the usual programming material, this book includes a chapter on modular program design, and one on program debugging, testing, documentation, and maintenance. Unlike some authors, they do not succumb to the illusion that structured programming means no errors in programs! That is what makes this book more than just another textbook. The technical material is well organized and presented, and it also contains a great deal of practical wisdom, which will save the careful reader from many unnecessary mistakes.

The Pascal language used here is that of the Jensen & Wirth text, *PASCAL Users Manual and Report,* Springer-Verlag, 1974. It is presented in syntax graphs inline in the text as needed, and complete in an appendix. The timing of the text may be unfortunate. It is a rewrite of a 1978 edition. At that time, there was no official standard for Pascal, so it contains no reference to ISO standards or to conformant array parameters. That is not a serious problem for an introductory text, because the core of the language is almost unchanged since 1978. There is practically no reference to UCSD Pascal or other non-standard versions. There is a brief discussion of versions with INTERACTIVE files (which do not do a GET first in READ), and a note about versions with direct access files.

The only thing missing from this excellent text is a discussion of quicksort, which other authors have managed to present in an elementary manner. Let's not produce another generation of programmers who know only bubble sort!

Language Translations

By John Zarrella
Published by Microcomputer Applications, Inc.
$14.95, 200 pages
Reviewed by Michael Carter

Not the least of the advantages of the microcomputer revolution is the encouragement and indeed the incentive it gives the owner of a personal computer to delve into its workings and operations. This will inevitably lead to the spread of computer know-how from a select few to a wider audience – a development which I for one heartily applaud. It also implies the existence of a market for introductory computer texts, of which this book is a good example. The author of such a book faces a considerable problem – how to aim at an easy presentation without oversimplification. The current author is partially – but only partially – successful.

No user of a personal computer can avoid forming an intimate acquaintance with at least one language translator (most commonly BASIC). A language translator is a computer program which accepts a human-oriented source language as input, translating it into a form which can be directly executed by the computer. It is the medium through which the user conveys his or her wishes to the machine.

After a general introduction, Zarrella deals in turn with the three main types of language translator: assemblers, interpreters (e.g., APL, BASIC, Forth), and compilers (e.g., C, COBOL, Pascal, Fortran). He appropriately emphasizes two important points. First, interpreters and compilers are not distinct categories but rather the ends of a continuum with most language translators lying somewhere in-between. The reader will thus understand why a BASIC program is stored in the machine in a form which he or she cannot readily read. The second important point is that the basic tasks required of a language translator are the same whether it is an assembler, compiler, or interpreter. Consequently, assemblers, compilers, and interpreters are made up of essentially the same functional components.

Zarrella makes convenient use of the latter point in the second half of the book when he deals in successive chapters with the functional components of any language translator: lexical analysis, syntax analysis, semantic analysis, code generation, and optimization. The lexical analyser examines the source text and divides it into its component parts: symbols, numbers, punctuation, and so on. The syntax analyser parses the input program to verify that it is a valid program in the language and to provide an appropriate framework for the next step. This is semantic analysis which determines the meaning of the source program. Code generation produces the actual executable machine code. Finally, optimization makes minor changes in the executable machine code to increase execution speed and/or decrease storage requirements. It should not be concluded that these five components of a language translator can necessarily be distinguished as separate entities that follow one another in the order given. They are conceptual divisions, and with the exception of optimization, they must be provided for in every language translator. The five constituent parts provide an appropriate framework for approaching the general task of language translation.

The five chapters dealing with the constituent parts of a language translator

are preceded by a chapter on "Languages and Grammars." This is the theoretical core on which language translation, in particular syntax analysis, is based. Unfortunately, this is not easy material to grasp and the chapter will probably require more than one reading. It is here and in the chapter dealing with syntax analysis that Zarrella's exposition falters. I think this is because he is trying to compress some complex ideas into two short chapters. Unfortunately, this material is really the core of the matter and it left me feeling unsatisfied. I think some more material would have been justified. Further, I found his use of the analogy of a divisionalized company to explain top-down parsing overdone and superfluous.

There are three supplementary chapters which have not yet been mentioned. One deals with program development utilities: linkers, loaders, library managers, etc. A second deals with macros, which are most commonly (but not exclusively) found in conjunction with assemblers. A third deals with the important topic of error processing. A glossary, a short bibliography, and an index round out the book.

This book has an admirable objective: to present the general concepts of language translation, emphasizing the similarities between the different classes of assembler, interpreter, and compiler in a manner accessible to the neophyte. This is a very ambitious task and one in which the author is not entirely successful. I suspect that it is more likely to whet the appetite than satisfy it. Hopefully, it will encourage the reader to seek further information in one of the more substantial texts. To any computer user who is curious about what lies behind the cosy facade of a BASIC interpreter or wonders why Pascal can't be like BASIC, I suggest this book as a good place to start.

Z80 Assembly Language Subroutines

By Lance A. Leventhal and Winthrop Saville
Published by Osborne/McGraw-Hill
$15.95, 498 pages, paperback
Reviewed by Paul J. Gans

Learning assembly language for a new computer is not easy for anyone. Even experienced programmers go through a period of adjustment when faced with a new machine. Novices are urged to study the architecture of the new machine to learn its op codes and addressing modes and to struggle as best they can to write simple routines that have been written countless times before.

More experienced programmers know that there is more to it than that. They do not become comfortable in the machine language of a new machine until

they have mastered its idioms. *Z80 Assembly Language Subroutines* is a book of idioms for the Z80. It is a wonderful book that can be read with profit by anyone not yet able to speak idiomatic Z80.

The book begins with an explanation of Z80 architecture (with a section for experienced programmers, pointing out distinguishing features and difficulties with the Z80) and the Z80 instruction set. The instruction set is presented from a programmer's point of view. Thus instructions are grouped according to function. For instance, under "Storing Registers in Memory" follow direct storage of registers, indirect storage of registers, indexed storage of registers, etc. The instruction set is covered fully with many examples.

Chapter 2, entitled "Implementing Additional Instructions and Addressing Modes," is filled with instances of common operations requiring more than a single instruction. These operations occur repeatedly in real programming. The novice must, each time, figure out how best to perform them. These operations are, in fact, idioms, and are at the fingertips of experienced programmers. For example, how does one compare the contents of the HL register pair to a 16-bit immediate value? On page 81, Leventhal and Saville give two methods, one requiring two instructions, the other, three. Chapter 3 is devoted to common programming errors and includes such topics as using the flags incorrectly, confusing registers and register pairs, and handling arrays incorrectly.

The bulk of the book, however, is devoted to assembly language subroutines. There are more than 50 of them, covering topics such as code conversion (binary to BCD, ASCII decimal to binary, EBCDIC to ASCII, etc.), array manipulation, arithmetic (multiple-precision binary division, etc.), string manipulation (compare, concatenate, etc.), array operations, input/output, and interrupts. Each subroutine listing is elaborately commented, entry and exit conditions are clearly given, and execution times (in machine cycles) are provided. There is an introductory explanation of the function of each subroutine and a description of how it works. Best of all, test data and results are given. Much can be learned by examination of these subroutines, and, better yet, they are useful.

The Z80 instruction mnemonics used are Zilog's, which I believe are superior to the multiple and incompatible sets of pseudo-8080 mnemonics in common use. All notational conventions are explained. Appendices contain a detailed Z80 instruction set summary with reproductions of the Zilog instruction charts, a programming reference to the Z80 PIO, and an ASCII character set chart. The book concludes with a 23-page glossary and a useful index.

Mathematical Theories of Optimization; Proceedings, S. Margherita Ligure 1981; Lecture Notes in Mathematics Series No. 979

Edited by A. Dold and B. Eckmann
Published by Springer-Verlag
$14.50, 268 pages, paperback
Reviewed by Robert Irving

Optimization is the science of obtaining the maximum return for investment of resources under a given set of conditions. The field is also known as "operations research" from its initial application in World War II. The most common applications lie in the areas of inventory control, product selection, and manufacturing scheduling.

Engineers and computer scientists in the field of optimization are usually looking for applied solutions to real problems. They generally want either an algorithm or a program which will solve the problem at issue. Mathematicians, on the other hand, are not necessarily looking for solutions as such. They are more interested in the nature of the statement of the problem. Such things as "existence theorems"

(does a solution exist?) and "robustness" (does the solution apply over a broad range of conditions?) are of more import than a particular solution.

This book is pure mathematics with potential practical application. Consequently, it is of little direct interest to most computer programmers.

Ultimately, some of the articles may lead to algorithms of great importance. For example, the article by L. Cesari on "Existence of Solutions and Existence of Optimal Solutions" may someday encourage other mathematicians to produce algorithms with which we can optimize some problems not now covered, but possible.

On the other hand, I see little to no potential use for a discussion by A. Bensoussan on page 71, entitled "On the Production Smoothing Problem," in which he extends a discrete case to a continuous case. First, I know of no "continuous" production process. Even oil refineries, which work with large volumes of material, operate on a "batch" basis. No two batches have the same characteristics, hence the run will be "continuous" only for the length of the batch. Second, all of us who make mathematical models on the computer know that we are dealing with discrete processes, since that is the only kind of process the computer can deal with.

Unless you are a "pure" mathematician interested in the rigors of that field, I suggest you forgo buying this volume.

Teletext and Videotex in the United States

By John Tydeman, Hubert Lipinski, Richard P. Adler, Michael Nyhan, and Laurence Zwimpfer
Published by McGraw-Hill
$30.00, 314 pages
Reviewed by Susan Bowers

Teletext (one-way information services) and Videotex (two-way information services) are new technologies created by the marriage of computing and communications. They emerged in England and France in the 1970's, have been growing rapidly in Europe, Canada, and Japan, and are currently becoming important tools in this country.

Teletext and Vidotex in the United States is a look at Teletex and Videotex from a futurist viewpoint. Using careful research and documentation, this in-depth overview deals with three classes of services: public data bases, closed user groups, and private systems. Full of charts and tables, the book looks at the present and future states of this technology in the United States. The first five chapters deal with defining the phenomenon, detailing its history, assessing the current state of information services, and discussing possible future applications. The rest of the book develops a model which studies the possible extent and impact of Teletext and Videotex in the United States by the end of this century.

The book emphasizes such public policy concerns as the effect of those technologies on society, legal aspects, security, privacy, standards, consumer protection, international ramifications, advertising liability, and impartiality. The modeling and forecasting are organized and carefully done, covering the possible futures from several perspectives. There is an impressive appendix on policy issues and an extensive bibliography, as well as a comprehensive index.

Teletext and Videotex in the United States provides an interesting look at future trends in the information services. Several parts of it would add much to a study of computers in society; it could also be an excellent textbook for a course on the future impact and direction of information services in the United States.

DDJ

by Michael Wiesenberg

Stop Noise

Kalglo's **IPC** international model computer power control console with surge suppression and power line noise filter has six individually switched IEC-320/CEE-22 outlets in dual separately filtered banks of three each. It absorbs 318 joules at 18,200 amp pulse, is rated at 10 amps maximum with 100 or 260 VAC operation, 50 or 60 Hz, weighs 2.5 pounds, and costs $150. **Reader Service No. 101.**

The **Lemon AC Surge Protector** from Electronic Protection Devices gives protection in excess of 6000 volts and 200 amps, with instantaneous voltage clamping including grounding to prevent flashover to other ports and damage to equipment. The company thinks its product is so good that insurance by Lloyd's of London is included, protecting user hardware up to $2500 from damage caused by transient voltage surges. The Lemon lists for $59.95. **Reader Service No. 103.**

Buffer It

Big-Buffer from Mikrocomputertechnik of Germany is a Centronics-compatible hardware spooler for parallel printers. The 8K version is $170, the 32K version is $195, the 64K is $256, the 96K is $317, and the 120K is $363. You get all cables and connectors (36-pin Centronics with female input and male output). Power comes from the printer using pin 18 of the Centronics connector, or you can purchase a separate 110V, 60Hz power supply for $30. These prices are based on the exchange rate at the time of the release, 2.60 DM to $1 US, and are subject to change. **Reader Service No. 105.**

Can't Rip It

Micro Format has **Flexi Disk Envelopes** made of Dupont's Tyvek, a supposedly tear- and wear-proof substance that looks and feels like paper. They included a sample with their press release, and I couldn't tear the stuff. Each package has 25 envelopes, and you get five packages postpaid for $25. **Reader Service No. 107.**

TRS-80 Assembly Language Programming

TRS-80/Z80 Assembly Language Library by Craig Lindley from WG-Books has 75 program listings to thoroughly explain assembly language on the TRS-80, and includes two TRSDOS disks with 33 source and object files, for $34.97. **Reader Service No. 109.**

Dr. Dobb, Meet Dr. Logo

Dr. Logo, from Digital Research, is an advanced version of Logo for 16-bit micros, in particular the IBM PC and XT. It uses turtle graphics, just like the popular version of the language. Written in C, Dr. Logo is transferable to other systems. You get a larger work space, with over 10,000 nodes of memory, built-in help commands, a full-screen procedure editor (with on-line help as well). Also supported are comments, indentation, user-defined words, windowing, upper and lower case, new primitives, string processing, line editing, and debugging with a trace mode and a watch mode. The latter is for checking or modifying variables or expressions immediately after execution of a statement. Cross referencing, double-precision floating point, transcendental functions, and a $149.95 price tag are part of the package as well. **Reader Service No. 111.**

Go Forth on PC

HSFORTH, from Harvard Softworks, is Forth-79 for the PC and other 8086 and 8088 machines. Word names, definition lists, and machine codes use separate segments, permitting programs up to 192K. HSForth runs the Sieve of Eratosthenes in 47 seconds, as opposed to 70 to 140 on other Forths, and 2000 for interpreted BASIC. Auto-Opt and the Micro-Asm extension reduce the time to five seconds. Even optimized machine code, according to Harvard Softworks, runs no faster than five seconds. $220. **Reader Service No. 113.**

Things I've Tried

Usually the products I write about in this column are picked from a stack of press releases sent each month to *DDJ*. I rarely have the product in front of me when writing about it, and descriptions here are not intended to be endorsements.

Once in a while I do get the opportunity to try out products, and I'd like to share a few with you.

You may have gathered from previous columns that I like inexpensive computers, particularly the Timex products. One of the wonderful things you can do with a TS1000 or 1500 is learn Z80 assembly language. BASIC on a Timex runs achingly slow, particularly since some 60% of the microprocessor's time is spent updating the screen. Machine code programs, however, run literally hundreds of times as fast as BASIC ones. Several good books have been written about machine code programming on the TS1000. The best of these are two by Dr. Ian Logan, whose knowledge of the ZX80 and 81 (predecessors of the TS1000) is exceeded only by that of Clive Sinclair (inventor of these and the Spectrum, a version of which should now be available, modified by Timex and called the 2068).

You cannot write effective code for these little wonders without knowing the entry points for the various system variables. Melbourne House, an international publishing company, has Dr. Logan's *Understanding Your ZX81 ROM*, $14.95, and *The Complete TS1000/Sinclair ZX81 ROM Disassembly*, $19.95, and their own apparently authorless *Machine Language Programming Made Simple for Your Sinclair & Timex TS1000*, $14.95. I have all three of these, and they are excellent. Melbourne also offers *Not Only 30 Programs for the Sinclair ZX81*, $9.95, program listings that ingeniously all run in the "unexpanded" TS1000 (2K RAM), and what appears to be the ultimate hardware manual for the two "simpler" machines, *The Ins and Outs of the Timex TS1000 & ZX81*, by Don Thomasson, $12.95. Add $2 for p.&h., and residents of California, Maryland, and Tennessee add sales tax. Melbourne carries software and books for Timex, TRS80, Vic 20, and Commodore 64. They will send you a free catalog and

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal

For Users of Small Computer Systems

P.O. BOX E
MENLO PARK, CA 94026

Dr. Dobb's Journal

For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date. Use only one card per person.

December 1983, No. 86

Name ____________ Address ____________

Phone (   ) ____________ City/State ____________ Zip ______

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81
82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108
109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162
163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189
Articles: 190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214		

Comments:

Purchase of Magazine:

1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ____________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal

For Users of Small Computer Systems

P.O. BOX E
MENLO PARK, CA 94026

Dr. Dobb's Journal

For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date. Use only one card per person.

December 1983, No. 86

Name ____________ Address ____________

Phone (   ) ____________ City/State ____________ Zip ______

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81
82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108
109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162
163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189
Articles: 190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214		

Comments:

Purchase of Magazine:

1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ____________

PLACE
STAMP
HERE
The Post Office
will not deliver
mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems

ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94026

Dr. Dobb's Journal
NOW AT BIGGER SAVINGS!

~~$35.40~~
$25.00

If you take advantage of this special offer you save over $10 off newsstand prices, that's a 30% savings!

_____Please charge my: ____Visa ____MasterCard ____American Express
_____Payment enclosed _____Bill me later

Card # ________________ Exp. date ________

Signature ________________________

Name ________________________

Address ________________________

City ____________ State _____ Zip _______ Z6

Offer good in USA only. Foreign rates upon request. Please allow up to six weeks for first issue. This offer good until March 31, 1984.

A publication of People's Computer Co., P.O. Box E, Menlo Park, CA 94026.

PLACE
STAMP
HERE
The Post Office
will not deliver
mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems

ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94026

Dear Reader, **December 1983, No. 86**

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond. Simply fill it out and drop it in the mail. We take care of the rest. Thanks for taking a few minutes to talk with us.

—Ed.

Which articles or departments did you enjoy the most this month? Why? (Please indicate order of preference.)

____

____

Comments or suggestions: ________________

____

____

Name ________________________

Address ________________________

____

are looking for contributions from authors. **Reader Service No. 115.**

Far better than hand-assembling your machine code programs and then trying to POKE them into various memory locations is to use an assembler. International Publishing & Software Inc. offers **ZX Assembler.** It has its own editor. You write programs using standard Z80 mnemonics. No need to figure jump addresses; just use labels and the assembler figures them for you. Programs assemble in seconds and execute either directly from ZX Assembler's monitor mode or from a program that executes with a USR statement where the code is in a REM statement created by the assembler. The assembler also retains the source code in a second REM statement.

You can SAVE the entire program on tape, and alter, modify, and reassemble it. Those familiar with machine code on the TS will appreciate this method, because code loaded in other areas is either difficult to save or cannot use relative addressing. In the monitor mode you can move whole blocks of code from one memory location to another, examine and modify any location of RAM, inspect and modify registers, and search the entire memory for any byte sequence. ZX Assembler costs \$14.95. International Publishing provides a free catalog and the name of the nearest dealer that handles this and other useful products, including **ZX Bug**, \$14.95, for debugging, editing, and running machine codes, **Toolkit**, \$14.95, to add nine powerful commands, including a renumber that changes all GOTO and GOSUB references, **Fastload**, \$19.95, to load any program four to six times as fast, **ZX Forth**, \$29.95, for the ease of BASIC with machine code speed, and a stack of clever games. **Reader Service No. 117.**

If I knew that a product did not live up to its claims, I would not list it here. Usually I have no way of knowing that, because I just go by the press releases, which, naturally, are never unfavorable. Such was the case with Memotech, whose various memory modules and auxiliary keyboard for the TS1000 I cited a few months back. I believed their hype myself and shortly afterward bought their keyboard and 32K memory for my system. That first 32K memory didn't work at all, and I took it back to the store. The replacement 32K refused to work in conjunction with either Memotech's keyboard or the Timex printer. This was not much use to me since I needed the printer for listings during program development. I took my entire system to the store, and *they* could get the 32K memory and auxiliary keyboard to work with their systems only separately, not together.

I called Memotech several times for help, and they kept insisting that they had tested a setup identical to mine with no problems. Finally they agreed to look at my system if I would send it to them and, if either of their products was defective, to replace or fix it. I called them when the system was in their hands, and the technician to whom I spoke said he had set up my system and it had been running perfectly for ten minutes, and he could find nothing wrong with it. But while I was on the phone with him he said, "Wait a minute. It's failing." He conceded that there "might" be a problem with the memory module, which Memotech replaced. They then sent back my system.

When the system returned, again I could make the 32K memory work by itself only, not together with either the Timex printer or Memotech's keyboard. The keyboard, too, only worked by itself. The store from which I bought the memory unit was kind enough to take it back in exchange for a Timex 16K memory (and even refunded the difference in price!), with which I have had no problems. I sent the keyboard back to Memotech, requesting a refund, citing the "free ten-day trial period" clause in their warranty, stating that, while the ten days had elapsed all the problems had begun the instant I began using the keyboard and that I had been in contact with them right from the start. They refused to give me a refund, claiming that the refund applied only to those who had purchased directly from them, but instead returned a replacement for my keyboard. If I wanted a refund, they told me, I would have to get it where I bought the keyboard.

The store, of course, did not want to give me a refund, because the 1500 would soon be released with its "real" keyboard, and auxiliary keyboards for Timex computers would soon be obsolete. They said that I should contact Memotech for a refund. (My guess is that Memotech is stuck with a warehouse full of these useless keyboards.) As I expected, the replacement worked no better than the original. At this point I no longer had their 32K memory, but the keyboard worked with neither Timex's printer nor the 16K memory.

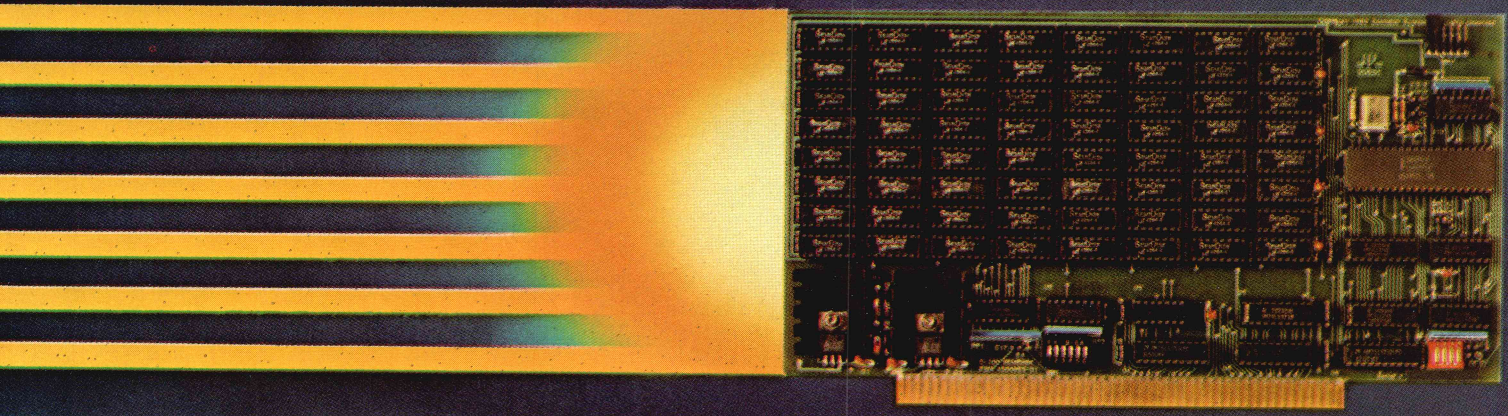
By this time I had heard from the South Bay Area Timex/Sinclair Users' Group (mentioned in my October column) that Memotech was offering a free "modification" to purchasers of their 32K memories. It seems several other people had discovered that *their* memories did not work with Timex printers. Yet all the time I had been in communication with Memotech they insisted there was no incompatibility between their products and Timex printers; furthermore, they had never heard of anyone having any problems like mine. *Altogether, I went through three 32K memory modules and two keyboards without finding even one I could use.*

Now *Dr. Dobb's* has received a press pack announcing Memotech's \$595 **MTX-512** World Class Computer™. It has a Z80A 4MHz processor with 80K standard RAM (expandable to 512K; 16K is dedicated to video). Among other things, it comes with Oxford BASIC and a language called NODDY, a real time clock, and a 79-key keyboard. I hope this product is better than the peripherals I couldn't use, and that others will encounter more competent service and better support and attitude than I did. I know *I'll* never get the chance to find out. **Reader Service No. 119.**

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 207.**

THE PRICE OF *FAST* WAS JUST SHATTERED!



256Kbyte SemiDisk™ $995

For more than a year, we've been making the most advanced disk emulator available for microcomputers. The one that's taken the "waiting" out of computing. Now, we have some more news that'll set the world on fire: A price cut! The NEW 256Kbyte board is only $995. And the 512Kbyte SemiDisks for the S-100 and TRS-80 Model II are $1495. (1Mbyte unit is $2350.) So, what are you waiting for?

The SemiDisk is the **ORIGINAL** single-board microcomputer disk emulator. It has a greater storage density than any other: 1 Mbyte per board! And we've been shipping them for over a year! We didn't do this with 'me too' engineering. Our products are true innovations, based on reliable technology and proven designs, without the need for custom components.

Floppies are ok for data transfer or long-term storage. But they fall far short as online storage. If you are using high level languages, spelling checkers, word processors, databases and other disk-intensive software, you know the price you are paying: time. Your productivity is going down the drain. The SemiDisk disk emulator will save time and increase your productivity.

Even better, Release 5.0 of the SemiDisk CP/M-80 installation software contains SemiSpool, an automatic printer buffer. No extra hardware is required; it's all in the software. Up to 8 Mbytes of buffer space! It's a better solution than a $350 64Kbyte printer buffer that wastes space on your desk. Send documents of almost any length to the printer at a very high speed, then continue using the computer immediately. No Waiting!

SemiDisk

It's the disk the others are still trying to copy.

SemiDisk Systems, Inc.

P.O. Box GG Beaverton, OR 97075 (503) 642-3100

NO WAITING

Call 503-646-5510 for CBBS®/NW, a SemiDisk-equipped computer bulletin board.

Circle **no. 91** on reader service card.

EXPECT A LOT *from* VEDIT

The Professional's Editor for Program Development, Word Processing, Source Code Translations

Widely acclaimed as an editor, VEDIT has evolved to be much more. Only VEDIT offers the combination of a versatile full screen editor integrated with a powerful command language based on TECO. For the first time you'll be able to perform complex, yet useful, text manipulations that are virtually impossible with other editors or word processors.

VEDIT is completely customizable - it easily configures to your favorite keyboard layout. Use any function or cursor key you wish. Plus, our total commitment to hardware support ensures that VEDIT will be perfectly matched to your individual needs and to any microcomputer you are ever likely to own.

VEDIT cuts programming time in half - with multiple file handling, macro capability and special features for Pascal, PL/1, 'C', Cobol, Assembler and other languages. And it can help with source code translations (example ZILOG to/from INTEL translator macros are included). A complete line of translators will be available by the year's end.

Word processing is a snap with word wrap, paragraph and print functions. Command macros free you from tedious search/replace operations. Hundreds of search/replace on dozens of files can be performed by VEDIT without waiting or intervening.

Go ahead and expect a lot from VEDIT. Its performance and our support will satisfy your most exacting needs.

VEDIT is available for virtually every 8080, Z80 or 8086 microcomputer made. To order please specify your computer, operating system and disk format.

VEDIT - Disk and Manual
8080, Z80 or IBM PC $150
CP/M-86 or MSDOS $195
Manual only $18

InfoWorld
Software Report Card

VEDIT 1.36

	Poor	Fair	Good	Excellent
Performance	☐	☐	☐	☑
Documentation	☐	☐	☐	☑
Ease of Use	☐	☐	☑	☐
Error Handling	☐	☐	☐	☑

Quotes from InfoWorld review by Tim Daneliuk of last year's version of VEDIT

'The performance of this product is nothing short of outstanding!'

'The ability to customize the editor is expecially useful.'

'VEDIT is a 'virtual' text editor. The amount of memory in your computer does not limit the size of the file you can edit.'

'VEDIT acts just as its advertising and documentation claim, with no unexpected errors or other problems. If anything, the advertising claims for this product are understated.'

'This is a well-conceived product. The many Visual-mode control and escape sequences make editing text simple, and the Command-mode features greatly simplify file handling and memory management.'

VEDIT's FEATURES:

- True Full Screen Editing
- Horizontal Scrolling
- Edit Files One Disk in Length
- Automatic Disk Buffering
- Compact (Only 16K) and Fast
- Display of Line and Column
- Set/Goto Text Markers
- 'Undo' Key to Restore Line
- Automatic Indent/Undent
- Adjustable Tab Positions
- Repeat Function Key
- Text Move and Copy
- 10 Scratchpad Buffers
- Load/Save Buffers on Disk
- Powerful Command Macros
- Directory Display
- Edit Additional (small) Files Simultaneously
- Insert Another Disk File
- Unlimited File Handling
- Recovery from 'Full Disk'
- Change Disks While Editing
- Word Wrap, Format Paragraph
- Simple Printing
- 150 Page Indexed Manual
- Startup Command File
- Menu Driven Installation
- Program CRT Function Keys
- Support Newest CRT terminals
- Support Smart CRT functions
- Flexible Memory Mapped Support
- Customizable Keyboard Layout

CompuView
PRODUCTS, INC.

1955 Pauline Blvd., Suite 200 Ann Arbor, Michigan 48103 (313) 996-1299

ADVERTISER INDEX

Announcing the Winners!

Questionnaires were mailed out to a random sampling of *Dr. Dobb's Journal* subscribers. Those who completed the questionnaire were eligible for a raffle. A list of the prize winners follows. Thanks to all of you for completing the questionnaire. This annual reader survey is a valuable tool in planning the coming year.

One DDJ Bound Volume Set

Jack R. Davis, Knoxville, TN

DDJ Bound Volume 6

M. L. Aitel, Herndon, VA
Arthur L. Greenberg, Yardville, NJ
John Lobenstein, Santa Clara, CA
Edward A. Malin, Peekskill, NY
A. F. Niessner, Jr., Boalsburg, PA
Wm. W. Richardson, Sherman, TX

Dr. Dobb's T-Shirt

Steve Brooks, Minot, ND
John W. Campbell, Bozeman, MT
Roger L. Cole, Elmhurst, IL
Norman Dresner, Severn, MD
Daniel J. Feiling, Lake Jackson, TX
Steven J. Flap, Winona, MN
Richard Gerth, Abilene, TX
Ron Herbst, Terre Haute, IN
Michael R. Jankovich, Salt Lake City, UT
William M. Kline, Miami, FL
A. Kontos, Houston, TX
Jeffrey K. Lines, Collinsville, IL
Roland V. Lupient, Mosinee, WI
G. McNevin, Richardson, TX
Michael McCarter, Philadelphia, PA
Steve Monroe, Herndon, VA
Jerry Owen, Hopewell, VA
Robert Pierazek, Riverside, CA
Daniel Scalici, Detroit, MI
Glen A. Spooner, Oceanside, CA
Larry D. Thompson, Ann Arbor, MI
Richard E. Thurmond, Whittier, CA
Robert J. Wallace, Raleigh, NC
Gordon White, Maquoketa, IA
David Whip, Reistertown, MD

Congratulations!

SAGE™ TECHNICAL BRIEFING

4 It's Fast In Any Language.

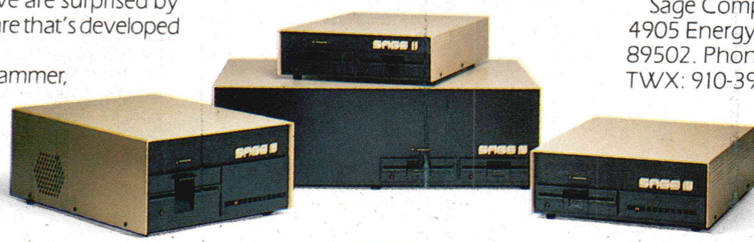
When it comes to software development, the difference between a Sage IV computer and other micros is like day and night.

With the Sage Computer it'll take you fewer days and nights to finish your program.

Speed is the reason. Speed resulting from the fast MC68000 microprocessor, fast architecture, fast operating systems, and blinding transfer rates.

The fact is, even we are surprised by the amount of software that's developed on Sage Computers.

So if you're a programmer, maybe you should spend some time learning how much time you could save using a Sage IV.



The Hottest Languages And Operating Systems.

Programs in nearly all of the important languages can be written on the Sage computer.

Included in the price is the **p-SYSTEM** operating system which supports **Pascal, BASIC,** and **FORTRAN.**

Other operating systems are optional. For **UNIX** fans there's **IDRIS,** which runs up to twice as fast (even without a Sage computer). **IDRIS** conforms to /USR/Group Standards Committee standards and programs written under it are highly portable to other micros. **CP/M** advocates please note that Digital Research has developed **CP/M-68K** for Sage hardware, providing a truly complete software development environment. Versions of **Pascal, BASIC, C-BASIC, C,** and **FORTRAN 77,** as well as a very fast **APL,** may be used under this operating system.

The Sage IV is also blindingly fast when running **hyperFORTH** with its extended programmer and user interfaces.

Besides **IDRIS,** other **Multi-User** operating systems that run on the Sage Computer are **PDOS, MBOS,** and **MIRAGE.**

A log of excitement has been brewing in the Pascal World over Niklaus Wirth's new **MODULA 2**—and it's available for Sage computers. So is **ADA,** for the record.

If you found your language here, you just found your computer. The 16-bit, 2-million operation/sec Sage IV micro with up to one MByte RAM and 18-MByte internal hard disk.

For more information and the name of your nearest SAGE dealer, call us today. And be sure to ask about our new soft ware catalog describing over 200 application programs for Sage computers.

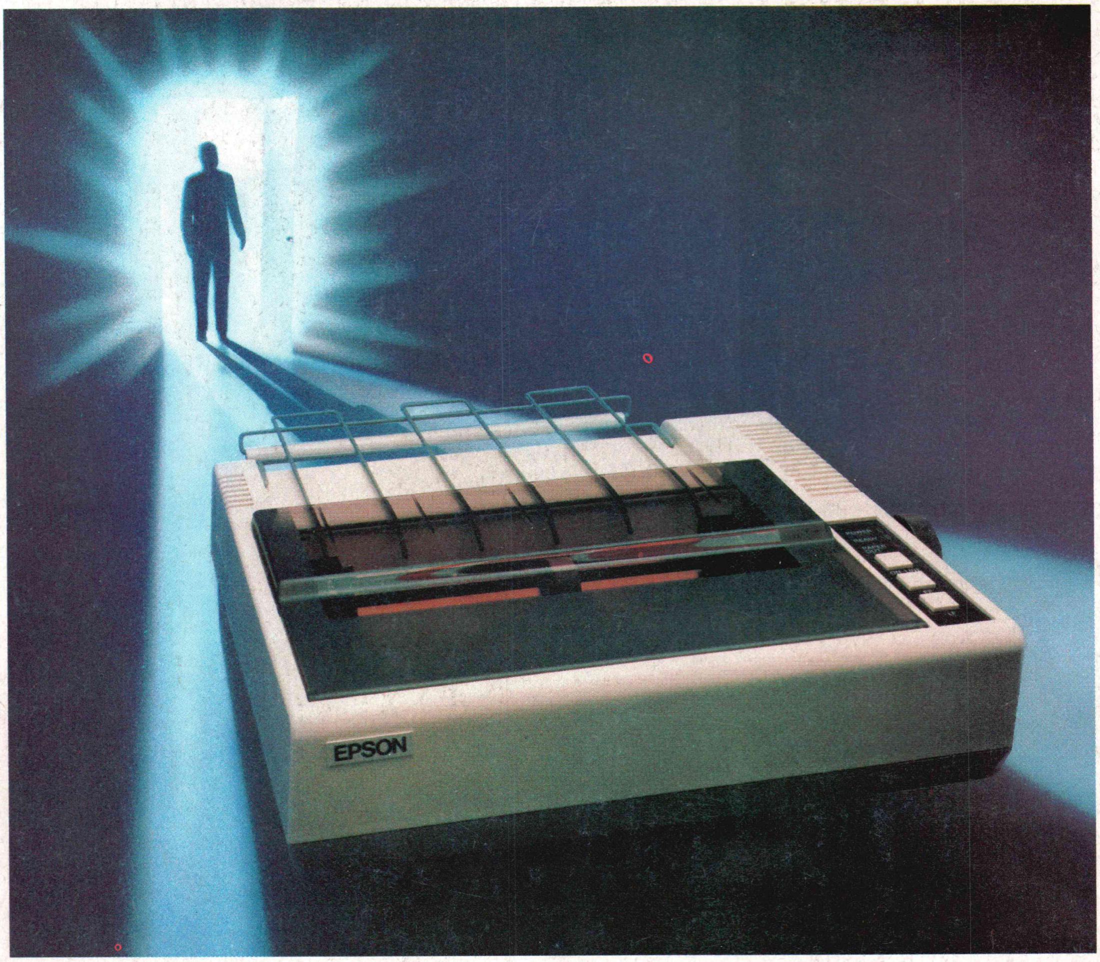
Sage Computer Corporate Office, 4905 Energy Way, Reno, Nevada 89502. Phone (702) 322-6868 TWX: 910-395-6073/SAGE RNO

Eastern United States
Sage Computer
15 New England
Executive Park
Suite 120, Burlington, MA
01803 (617) 229-6868

SAGE™ COMPUTER
The computer you don't wait for!

Circle **no. 90** on reader service card.

Introducing the new RX-80.



A printer this good could only come from one company.

Epson.

Take a look at the new Epson RX-80 with a critical eye and you'll know what we mean. Epson printers are put together with manufacturing tolerances in the micron range. And meticulously checked for quality — one at a time.

Our reliability rate is approaching perfection. Which means you'll probably never — ever — have a problem with your RX-80.

Frills at a no-frills price.
Just because the RX-80 is affordable doesn't mean we've left off the goodies. You get 128 different typestyles with a resolution so sharp you have to look twice to make sure it's dot matrix. You get bidirectional printing, logic seeking, and a speed of 100 characters per second. And you even get programmable forms control, graphics, and a special Quiet Mode to keep the noise down.

This is your printer.
No matter what computer you own — or will own — do yourself a favor and make sure your printer is an Epson RX-80.

We build them to be the finest printers on earth. No matter how small the price.

Number one.
And built like it.

EPSON
EPSON AMERICA, INC.
3415 Kashiwa Street, Torrance, CA 90505

Please call (800) 421-5426 for the dealer nearest you. In California (213) 539-9140

Circle no. **38** on reader service card.